Trajectory Sampling Value Iteration: Improved Dyna Search for MDPs

(Extended Abstract)

Yicheng Zhou
Soochow University
zyc9012@163.com

Quan Liu
Soochow University
quanliu@suda.edu.cn

Qiming Fu
Suzhou University of Science and Technology
fqm_1@126.com

Zongzhang Zhang
Soochow University
zzzhang@suda.edu.cn

ABSTRACT

Traditional online learning algorithms often suffer from the lack of convergence rate and accuracy. The Dyna-2 framework, combining learning with searching methods, provides a way of alleviating the problem. The main idea behind it is to execute a simulation-based search that helps the learning process to select better actions. The search process relies on a simulated model of the environment that is built during learning. However, the model is not fully used in Dyna-2. To provide better solution quality, our paper improves the algorithm by applying value iteration, a model-based dynamic programming algorithm, to the search process with a trajectory sampling approach (DynaTSVI). Trajectory sampling is used to reduce high time complexity caused by dynamic programming. Experimentally, we use the Dyna Maze and the Windy Grid World tasks to analyze the proposed method in several aspects. Our results show that DynaTSVI outperforms Dyna-2 in both deterministic and stochastic environments in terms of convergence rate and accuracy.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Control theory, Dynamic programming*

General Terms

Algorithms, Experimentation

Keywords

Reinforcement learning, Dynamic programming, Trajectory sampling, Model building

1. INTRODUCTION

The Dyna structure [3] is a reinforcement learning technique that combines learning with planning. It builds a model by memorizing history samples during learning and uses Q-learning to update the value function. Then in planning, the value function is updated again using randomly chosen samples from history. The planning process reuses the samples in order to speed up the convergence.

Recently, a newly proposed Dyna structure, Dyna-2 [2], has achieved success in the game Go. The algorithm separates the learning and planning into individual procedures. For learning, it uses the TD(λ) algorithm to get samples from the real environment and improve the policy. For planning, it uses the proposed TD search algorithm to exploit the experiences from the learning process.

To separately represent policies produced from the two processes, Dyna-2 maintains a long-term value function Q and a short-term value function $\bar{Q}$. Here, Q represents the approximated policy of the learning problem, and $\bar{Q}$, acting as an action selector for the learning process, represents the local knowledge of the sub-MDP problem staring from the current state. Compared with Monte Carlo tree search (MCTS) [1] algorithms, TD search provides better generalization and leads to better search quality.

The search process is based on a simulated model built during learning. However, a drawback of Dyna-2 is that, given the simulated model, the search process simply interacts with the model rather than use the actual state transition probabilities and reward distributions inside it. As a result, the search process may not be so efficient as it can be. Potential improvements still exist. Therefore, we propose the DynaTSVI algorithm that improves Dyna-2 on the search process.

2. NEW DYNA SEARCH

We introduce value iteration, a dynamic programming algorithm, to the search process. Value iteration is model-based, and its major advantage is the ability to produce precise solutions. In this way, the model built in Dyna-2 can be better used. However, directly using standard dynamic programming confronts the following two issues: Firstly, dynamic programming works on complete model, but during searching, the simulated model is refined gradually. Thus, the early models do not satisfy the requirement. Secondly, even though the model is complete, dynamic programming requires full backups on the entire state-action space, which

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May, 4–8, 2015, Istanbul, Turkey.*

Algorithm 1 Trajectory sampling value iteration

Initialize Q arbitrarily, e.g., $Q \leftarrow 0$
repeat for each episode
  $x \leftarrow x_0$
  $u \leftarrow \epsilon$-greedy$(x; Q)$
  repeat for each step of episode
    Execute u, obtain r and x'
    $Q(x,u) \leftarrow \sum_{\bar{x} \in X} f^u_{x\bar{x}} \left[\rho^u_{x\bar{x}} + \gamma \max_{\bar{u} \in U} Q(\bar{x}, \bar{u}) \right]$
    $u' \leftarrow \epsilon$-greedy$(x'; Q)$
    $x \leftarrow x', u \leftarrow u'$
  until x' is terminal state
until Q is satisfactory

leads to much higher time complexity compared with TD methods. When the state-action space is large, dynamic programming with full backups is not feasible.

To overcome the problems, we apply a trajectory sampling approach to the value iteration algorithm, as shown in Algorithm 1, where $f^u_{xx'}$ is the state transition function and $\rho^u_{xx'}$ is the reward function. Rather than updating the values of all states, the idea of our approach is to update the values only on the states that have been visited during the episode (i.e. trajectory). In this way, the high time complexity of dynamic programming can be reduced. Compared with TD algorithms, given a model, this approach may provide better approximations to the true results without the loss of efficiency.

We construct the simulated model by maintaining a sample list $L(x,u)$ for each state-action pair. When a sample (x,u,r,x') is obtained from the learning process, the experience is added to the corresponding state-action pair, i.e., $L(x,u) \leftarrow r, x'$. When the search process samples from the model, a randomly chosen experience (r,x') from $L(x,u)$ is given, $r, x' \leftarrow \textbf{random}(L(x,u))$. The distributions of the samples in the lists implicitly represent the state transition probabilities. The model functions are computed by:

$$f^u_{xx'} = \frac{N[L(x,u)|L(x,u) = x']}{N[L(x,u)]}, \quad \rho^u_{xx'} = r,$$

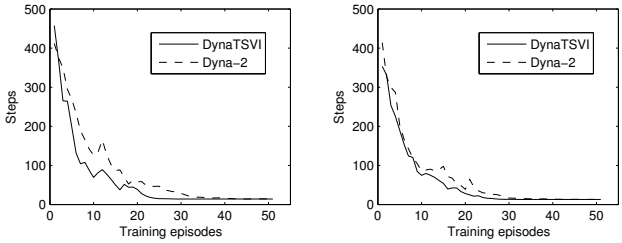
where N is a function that gets the number of elements in the list under given conditions.

We apply the trajectory sampling approach of value iteration to the search process and propose the DynaTSVI algorithm. Before selecting an action, the agent executes several searches (n times) on the current state. When search is done, the short-term value function $\bar{Q}$ is able to evaluate the current state relatively accurately. Then, the agent uses the short-term value function to select an action u_t following an ϵ-greedy policy. After obtaining a sample from the environment, the TD(λ) algorithm is applied to update the long-term value function Q. At the same time, the agent updates the model using the new experience.

Note that, with the employment of dynamic programming, both value functions are represented in tabular form. Therefore, DynaTSVI works on environments with discrete state spaces.

3. EXPERIMENTS

We evaluate DynaTSVI with two classic reinforcement learning problems: the Dyna Maze and the Windy Grid World. Specifically, the former experiment is in a deterministic environment and the latter is in a stochastic one. Our goal is to compare the convergence rate of the two algorithms.



(a) DynaMaze (b) Windy Grid World

Figure 1: Comparisons of convergence rate.

In both experiments, the maximal episode step is set as 2000. An episode ends when the agent reaches the goal or the number of steps taken has exceeded 2000. The search count n is set as 10 by default. Other parameter settings are: $\alpha = 0.1$, $\gamma = 0.95$ and $\epsilon = 0.1$.

The convergence curves of both experiments are shown in Figure 1. It can be observed that on Dyna Maze, DynaTSVI converges faster than Dyna-2. The former converges around 25 episodes of training, while the latter takes around 35, revealing the efficiency on execution of DynaTSVI. On Windy Grid World, DynaTSVI converges 5 steps earlier than that of Dyna-2 in average. In addition, after several runs on both algorithms, there is a 98% chance that DynaTSVI can converge to the optimal policy, while on Dyna-2, the percentage drops to 83%. This tells that DynaTSVI has higher convergence accuracy.

4. CONCLUSION

In this paper we show that the search process of Dyna-2 can be improved by value iteration, using a trajectory sampling approach. It is adapted to both deterministic and stochastic environments. Experiments on the Dyna Maze and the Windy Grid Word tasks have indicated that DynaTSVI has higher convergence rate and accuracy than Dyna-2.

Future works on this topic may include modeling environments with continuous state space. Thus, it is possible to apply our approach to more complex and realistic problems.

Acknowledgments

This paper is supported by National Natural Science Foundation of China (61272005, 61472262) and Natural Science Foundation of Jiangsu (BK2012616).

5. REFERENCES

[1] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*, pages 72–83. Springer, 2007.

[2] D. Silver, R. S. Sutton, and M. Müller. Temporal-difference search in computer go. *Machine Learning*, 87(2):183–219, 2012.

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