Computational Details of Linear Dyna Algorithms on Mountain-car

Hengshuai Yao
Department of Computing Science
University of Alberta
Edmonton, AB, Canada T6G2E8

Shalabh Bhatnagar
Department of Computer Science
and Automation
Indian Institute of Science
Bangalore, India 560012

Dongcui Diao
School of Economics and Management
South China Normal University
Guangzhou, China 518055

Abstract

This supplementary material contains the computational details of linear Dyna algorithms on Mountain-car. The matrix F is very large and relatively dense, *e.g.*, $30,000^2$ for state-action model. This leads to very slow online performance. We avoided the computation of F, and used a least-squares computation of projection.

1 Least-squares Computation of Projection

Gradient descent estimation of F and projecting directly using F lead to a very slow online performance because F is not sparse, although $\Phi^T DP\Phi$ and $\Phi^T D\Phi$ are both sparse. Notice that we only need F for projection operation. In our experiment of Dyna(k)-lambda (and linear Dyna), we applied a least-squares computation of the projection, which makes use of Matlab backslash operator.

In particular, a projection $L^{(k)}\tilde{\phi} = (\lambda\gamma)^{k-1}\gamma F_t\tilde{\phi}$ was computed by decomposing into

$$\begin{aligned} F_t\tilde{\phi} &= (\Phi^T P_t^T D_t \Phi)(\Phi^T D_t \Phi)^{-1}\tilde{\phi} \\ &= G_t \cdot (E_t^{-1}\tilde{\phi}), \end{aligned}$$

where G_t and E_t are very sparse, and updated every time step by

$$G_{t+1} = G_t + \vec{\phi}_{t+1}\phi_t^T,$$

and

$$E_{t+1} = E_t + \phi_t\phi_t^T.$$

First, $E_t^{-1}\tilde{\phi}$ is computed by Matlab backslash operator, "$E_t \backslash \tilde{\phi}$", and then the result is left multiplied by G_t. Both operations can take advantage of the sparsity. For linear Dyna with state features, projection using F_a was also computed similarly. For linear Dyna, f is also computed by the backslash operation

$$f = E_t \setminus b_t,$$

where b is updated by

$$b_{t+1} = b_t + \phi_t r_t.$$

Further, we do not have to compute $l^{(k)}$ explicitly because it is only used in generating the simulated reward:

$$\begin{aligned}\tilde{r}^{(k)} &= (l^{(k)})^T\tilde{\phi}\\ &= (f^{(\infty)} - (L^{(k)})^T f^{(\infty)})^T\tilde{\phi}\\ &= (f^{(\infty)})^T\tilde{\phi} - (f^{(\infty)})^T(L^{(k)}\tilde{\phi}),\end{aligned}$$

where $(L^{(k)}\tilde{\phi})$ is already computed by the least-squares projection. ($f^{(\infty)}$ is also computed by backslash operation using the LSTD rule.)

The two tricks make projection much more efficient, and greatly reduce CPU time per time step of Dyna algorithms.

All E matrices were initialized to I, and all G matrices were initialized to 0 for linear Dyna algorithms in Mountain-car.