Linear Least-squares Dyna-style Planning*

Hengshuai Yao HENGSHUA@CS.UALBERTA.CA
Department of Computing Science
University of Alberta
Edmonton, AB, Canada T6G2E8

Abstract

World model is very important for model-based reinforcement learning. For example, a model is frequently used in Dyna: in learning steps to select actions and in planning steps to project sampled states or features. In this paper we propose least-squares Dyna (LS-Dyna) algorithm to improve the accuracy of the world model and provide better planning. LS-Dyna is a special Dyna architecture in that it estimates the world model by a least-squares method. LS-Dyna is more data efficient, yet it has the same complexity with existing linear Dyna that is based on gradient descent estimation of the world model. Furthermore, the least-squres modeling is computed in an online recursive fashion and does not have to record historical experience or tune a step-size. Experimental results on a 98-state Boyan chain example and a Mountain-car problem show that LS-Dyna performs significantly better than TD/Q-learning and the gradient-descent linear Dyna algorithm.

1. Introduction

Reinforcement learning (RL) features limited experience of the world, and there has been a demand for algorithms that use data efficiently. Classical Temporal Difference (TD) is not data efficient because each transitioning experience is used only once. A class of data efficient algorithms are known as the least-squares methods, such as least-squares TD (LSTD) (Bradtke and Barto, 1996; Boyan, 2002; Xu, He, and Hu, 2002), least-squares policy evaluation (LSPE) (Bertsekas, Borkar, and Nedic, 2004), incremental LSTD (iLSTD) (Geramifard, Bowling, and Sutton, 2006; Geramifard, Bowling, Zinkevich, and Sutton, 2007), and preconditioned TD algorithms (Yao and Liu, 2008), etc. However, the advantage of these methods is only known for fixed-policy evaluation while practical RL tasks are more challenging in that the policy changes from time to time.

Dyna architecture is another approach to reuse experience (Sutton, 1990; Sutton and Barto, 1998). Dyna builds a model of the world from online experience and uses the model for planning. Through planning, Dyna outputs an improved value function for acting and learning, which in turn gives the world model better experience for refinement. The world model thus projects states into more accurate future states and rewards and simulates more realistic experience. As an integrated architecture of circulating among learning, modeling and planning, Dyna is continuously self-improving, getting better and better experience, world model, planning, and value functions. Dyna was recently extended to linear function

*. Technical Report TR11-04, Department of Computing Science, University of Alberta

approximation to handle problems with a large state space (Sutton, Szepesvári, Geramifard, and Bowling, 2008). The key component of linear Dyna is a compressed linear model estimated from real world experience. The compressed world model is composed of a set of action models, each action model telling the results of taking the action at a given feature.

As the name suggests, the key component of model-based reinforcement learning is the world model. If we can improve the world model, model-based algorithms can give us better action selection, projection and planning. For example, the world model is used three times in linear Dyna: for *action selection* in both learning and planning, and for *projection* of sampled features in planning. In this paper, we focus on linear Dyna-style planning. Some used a multi-step model of the world for linear Dyna and their results show that the performance significantly outperforms that of a single-step model (Yao, Bhatnagar, and Diao, 2009). Moreover, experimental results show that the gradient descent based linear Dyna is sensitive to the step-size of estimating the world model (Sutton et al., 2008; Yao et al., 2009).

Our new algorithm, called *least-squares Dyna (LS-Dyna)*, is based on linear Dyna-style planning and can work with large-state-space problems. LS-Dyna improves the data efficiency of estimating the single-step model using least-squares approach. The single-step model of linear Dyna is composed of a matrix and a vector. A recursive least squares of the vector can be $O(n^2)$ in a similar way to value function approximation (Bradtke and Barto, 1996; Xu et al., 2002). What is the complexity of estimating the model matrix using recursive least-squares then? Because of the dimension is squared, it is tempting to think that it requires as high as $O(n^3)$. In this paper, we show that this is not true. A recursive least squares of the model matrix can be $O(n^2)$, as low as the model vector. Thus LS-Dyna does not have to pay more price for data efficiency. Furthermore, the online recursive least-squares method does not have a step-size that requires tuning, and thus LS-Dyna is much easier to use for practitioners of RL.

2. Background

2.1 Linear Function Approximation and TD(0) Algorithm

Given a Markov Decision Process (MDP) with a state space $\mathbb{S} = \{1, 2, \ldots, N\}$, the problem of *policy evaluation* is to predict the long-term rewards of a policy π for every state $s \in \mathbb{S}$ (Sutton and Barto, 1998):

$$V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t r(s_t, s_{t+1}), \quad 0 < \gamma < 1, \quad s_0 = s,$$

where $r(s_t, s_{t+1})$ is the reward received by the agent at time t. Given n ($n \leq N$) feature functions $\varphi_j(\cdot) : \mathbb{S} \mapsto \mathbb{R}$, $j = 1, \ldots, n$, the feature of state i is $\phi(i) = [\varphi_1(i), \varphi_2(i), \ldots, \varphi_n(i)]^T$. The idea of linear function approximation is to approximate V^π by $\hat{V}^\pi = \Phi\theta$, where θ is the weight vector, and Φ is the feature matrix whose entries are $\Phi_{i,j} = \varphi_j(i)$, $i = 1, \ldots, N$; $j = 1, \ldots, n$. At time t, TD(0) adjusts the weights by (Sutton, 1988)

$$\theta_{t+1} = \theta_t + \alpha_t(r_t + \gamma\theta_t^T\phi_{t+1} - \theta_t^T\phi_t)\phi_t,$$

where α_t is a positive step-size, ϕ_t corresponds to $\phi(s_t)$, and r_t is short for $r(s_t, s_{t+1})$.

2.2 The Gradient Descent Linear Dyna-style Planning

For policy evaluation linear Dyna builds a compressed linear model of policy π in the feature space: a transition model, $F^\pi \in \mathcal{R}^{n\times n}$, and a reward model, $f^\pi \in \mathcal{R}^n$. The transition model and the reward model are estimated using gradient descent in linear Dyna (Sutton et al., 2008):

$$F^\pi_{t+1} = F^\pi_t + \beta_t(\phi_{t+1} - F^\pi_t\phi_t)\phi_t^T, \tag{1}$$

and

$$f^\pi_{t+1} = f^\pi_t + \beta_t(r_t - f^{\pi T}_t\phi_t)\phi_t, \tag{2}$$

respectively, where the features and reward are all from real world experience, and β_t is a step-size.

Dyna repeats some steps of planning in each of which it proposes a sampled feature $\tilde{\phi}$ (we use $\sim$ to denote a simulated experience). The next feature is projected from the original one, *i.e.*, $\tilde{\phi}^{(1)} = F^\pi\tilde{\phi}$. The rewards leaving feature $\tilde{\phi}$ in one step correspond to $\tilde{r}^{(1)} = \tilde{\phi}^T f^\pi$. The imaginary experience is therefore $\tilde{\phi} \to (\tilde{\phi}^{(1)}, \quad \tilde{r}^{(1)})$. Dyna treats this imaginary experience as if it really happened, and applies TD(0) learning to improve the policy:

$$\tilde{\theta} := \tilde{\theta} + \alpha(\tilde{r}^{(1)} + \gamma\tilde{\theta}^T\tilde{\phi}^{(1)} - \tilde{\theta}^T\tilde{\phi})\tilde{\phi},$$

where $\tilde{\theta}$ is the parameter θ in the planning step.

For control, linear Dyna maintains a model for each action, and the general linear Dyna control with gradient descent update of the model is shown in Algorithm 2 (Sutton et al., 2008). Gradient descent is known for its slowness in convergence and labor in tuning of the step-size, both of which influence the performance of linear Dyna.

In Section 3, we propose a least-squares method to estimate the world model of linear Dyna. In Section 4, we propose LS-Dyna algorithms for both policy evaluation and control. Section 5 presents empirical results and Section 6 concludes the paper.

3. Least-squares Estimation of the Model

3.1 The General Problem

For the time being we focus on estimating the model of a policy π. The general model estimation problem is as follows.

We have a data set of transitioning experience $\mathbb{D}_\pi = \{(\phi_i, \phi_{i+1}, r_i), i = 0, 1, \ldots, d\}$, where experience are collected by following policy π. From the set $\mathbb{D}_\pi$, we would like to estimate a linear model (F_{d+1}, f_{d+1}), s.t.,

$$F_{d+1} = \arg\min_{F\in\mathbb{R}^{n\times n}} \sum_{i=0}^{d} ||\phi_{i+1} - F\phi_i||^2,$$

and

$$f_{d+1} = \arg\min_{f\in\mathbb{R}^n} \sum_{i=0}^{d} ||r_i - f^T\phi_i||^2.$$

In previous linear Dyna they used gradient descent algorithm to estimate the model (Sutton et al., 2008). In fact, this is a linear least-squares problem, and we can solve F_{d+1} and f_{d+1} directly using least-squares regression, which gives

$$F_{d+1} = \left[\sum_{i=0}^{d} \phi_{i+1}\phi_i^T\right]\left[\sum_{i=0}^{d} \phi_i\phi_i^T\right]^{-1}, \tag{3}$$

and

$$f_{d+1} = \left[\sum_{i=0}^{d} \phi_i\phi_i^T\right]^{-1}\left[\sum_{i=0}^{d} \phi_i r_i\right]. \tag{4}$$

3.2 Online Recursive Estimation

Solution (3) and (4) can be used for data set collected previously. However, we are interested in combining least-squares solution of the model with linear Dyna-style planning, meaning that we have to compute the least-sqares solution online. In the online setting, (3) and (4) can use Sherman-Morison for matrix inversion, which is $O(n^2)$, but (3) requires matrix product and is still $O(n^3)$. In the following we provide an online procedure to compute (3) in $O(n^2)$.

Suppose our current time step is t and the historical time steps are $h = 0, 1, \ldots, t$. At time step t we have seen t steps of transitioning experience and can estimate the model online using the linear least-squares (3) and (4). This computation method has to keep a record all historical transitioning experience. The following is online procedure without such a memory overhead.

Let $D_{t+1} = \sum_{h=0}^{t} \phi_h\phi_h^T$. We can accumulate D online by

$$D_{t+1} = D_t + \phi_t\phi_t^T.$$

Let $E_{t+1} = \sum_{h=0}^{t} \phi_{h+1}\phi_h^T$. Similarly we accumulate E online

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

We first compute a de-correlated feature

$$x_t = D_t^{-1}\phi_t, \tag{5}$$

where the matrix inversion is updated by Sherman-Morison formula

$$D_{t+1}^{-1} = D_t^{-1} - \frac{D_t^{-1}\phi_t\phi_t^T D_t^{-1}}{1 + \phi_t^T D_t^{-1}\phi_t}. \tag{6}$$

We can now compute matrix F using matrices D and E:

$$F_{t+1} = E_{t+1}D_{t+1}^{-1}.$$

Set $t = 0$; Initialize θ_0, F_0, f_0 and D_0^{-1}
Select an initial state s_0
for each time step t **do**
  Act according to the policy, observing s_{t+1}, r_t
  Apply TD(0):
    $\theta_{t+1} = \theta_t + \alpha_t(r_t + \gamma\phi_{t+1}^T\theta_t - \phi_t^T\theta_t)\phi_t$
  Update the model by recursive LS procedure (5)–(8)
  Set $\tilde{\theta}_0 = \theta_{t+1}$
  repeat for $p = 1, 2, \ldots, \tau$
    Sample a random basis unit vector $\tilde{\phi}_p$
    Project $\tilde{\phi}_p$:
      $\tilde{\phi}_{p+1} = F_{t+1}\tilde{\phi}_p$
      $\tilde{r}_p = \tilde{\phi}_p^T f_{t+1}$
    $\tilde{\theta}_{p+1} = \tilde{\theta}_p + \alpha_p(\tilde{r}_p + \gamma\tilde{\theta}_p^T\tilde{\phi}_{p+1} - \tilde{\theta}_p^T\tilde{\phi}_p)\tilde{\phi}_p$
  end
  Set $\theta_{t+1} = \tilde{\theta}_{\tau+1}$
end for

Algorithm 1: LS-Dyna algorithm for policy evaluation.

According to the accumulation of E and the Sherman-Morison update of D^{-1}, we have

$$\begin{aligned} F_{t+1} &= (E_t + \phi_{t+1}\phi_t^T)(D_t^{-1} - \frac{D_t^{-1}\phi_t\phi_t^T D_t^{-1}}{1 + \phi_t^T D_t^{-1}\phi_t}) \\ &= F_t + \phi_{t+1}x_t^T - \frac{F_t\phi_t x_t^T}{1 + \phi_t^T x_t} - \frac{\phi_{t+1}x_t^T\phi_t x_t^T}{1 + \phi_t^T x_t} \\ &= F_t + \left[\phi_{t+1} - \frac{F_t\phi_t}{1 + \phi_t^T x_t} - \frac{\phi_{t+1}x_t^T\phi_t}{1 + \phi_t^T x_t}\right] x_t^T, \end{aligned}$$

which can be simplified into

$$F_{t+1} = F_t + \frac{\phi_{t+1} - F_t\phi_t}{1 + \phi_t^T x_t} x_t^T. \tag{7}$$

In a similar manner to derive the rule for F_{t+1}, we have

$$f_{t+1} = f_t + \frac{r_t - \phi_t^T f_t}{1 + \phi_t^T x_t} x_t. \tag{8}$$

At each time step, a time order of estimating the model can be (5), (6), (7) and (8), of which the order of (6), (7) and (8) is not important.

The complexity of the algorithm is $O(n^2)$, which is the same as the gradient descent rule (1) and (2). It can be proved in a straightforward manner that the least-squares estimation of models converges with probability one to the same solutions as gradient descent estimation under similar conditions.

4. Linear Least-squares Dyna-style Planning

4.1 LS-Dyna for Policy Evaluation

In policy evaluation algorithm, learning and acting following the policy provides the transitioning experience for LS-Dyna. Given these transitioning experience, we use the recursive least-squares (LS) method developed in last section to estimate the world model of the policy. The other parts of LS-Dyna are the same as gradient descent linear Dyna. The complete algorithm of LS-Dyna for policy evaluation is shown in Algorithm 1.

There are interesting connections between LS-Dyna and least-squares policy evaluation algorithms. The methods all use least-squares methods: LS-Dyna uses least-squares methods to update the model, while LSTD, iLSTD and LSPE use least-squares methods to update the weight vector (the policy). For LSTD, the coefficients of a linear system (which can be viewed as a model) are estimated, and the weight vector is solved by least-squares at each time step. LS-Dyna builds the model F and b, but it does not solve the model at once [1]. Instead, the model is used incrementally, simulating experience and planning based on the simulation. In this sense, LSTD, iLSTD, and LSPE can be viewed as a batch method of using model or experience, while LS-Dyna is an incremental method.

4.2 LS-Dyna for Control

In control, the transitioning experience is provided online through learning and acting according to a changing policy. The data for estimating the world model is a set of transitioning experience following the actions that have been taken, $\mathbb{D} = \{(\phi_h, a_h, \phi_{h+1}, r_h), h = 0, 1, \dots t\}$. Notice that for control we estimate a set of action models rather than a single model in policy evaluation problem. In particular, at each time step, after taking an action, we update the model of the action by the action-dependent version of the recursive LS method, while the other action models are not updated. The other part of the new Dyna is the same as linear Dyna control algorithm. Finally the complete LS-Dyna for control is shown in Algorithm 2 (together with gradient descent linear Dyna).

Notice that Least-squares (LS) estimation of the model helps policy evaluation only in planning (for projection); however, besides projection in planning, LS estimation of the action models also helps action selection steps in both learning and planning.

5. Experimental Results

5.1 The 98-state Boyan Chain

We used a 98-state Boyan chain problem extended from the original 13-state problem (Boyan, 2002; Geramifard et al., 2007) using the same policy underlying the original Boyan chain. The root mean square error (RMSE) was computed at each episode to measure the performance of each algorithm:

$$(RMSE)_e = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(V(i) - \phi(i)^T\theta_e)^2},$$

1. One can solve in LS-Dyna for θ by $\theta = (I - \gamma F^T)^{-1} f$.

Set $t = 0$; Initialize $F_0^{\mathbf{a}}$, $f_0^{\mathbf{a}}$ (for all $\mathbf{a}$), and θ_0
Select an initial state s_0
for each time step t **do**
  Set $\phi_t = \phi(s_t)$
  Select an action:
    $a = \arg\max_{\mathbf{a}}\{\phi_t^T f_t^{\mathbf{a}} + \gamma\theta_t^T F_t^{\mathbf{a}}\phi_t\}$ (ϵ-greedy)
  Take a, observing s_{t+1} and r_t
  Set $\phi_{t+1} = \phi(s_{t+1})$
  Apply TD(0):
    $\theta_{t+1} = \theta_t + \alpha_t(r_t + \gamma\phi_{t+1}^T\theta_t - \phi_t^T\theta_t)\phi_t$
  Model estimate:
    gradience descent method (1)-(2) (action-dependent version), or
    recursive least-squares method (5)–(8) (action-dependent version)
  Set $\tilde{\theta}_0 = \theta_{t+1}$
  repeat for $p = 1, 2, \ldots, \tau$ //τ is the number of planning steps.
    Sample a random basis unit vector $\tilde{\phi}_p$
    Select an action:

$$\tilde{a} = \arg\max_a \left\{\tilde{\phi}_p^T f_{t+1}^a + \gamma\theta_{t+1}^T F_{t+1}^a \tilde{\phi}_p\right\}$$

    Project $\tilde{\phi}_p$:
      $\tilde{\phi}_{p+1} = F_{t+1}^{\tilde{a}}\tilde{\phi}_p$
      $\tilde{r}_p = \tilde{\phi}_p^T f_{t+1}^{\tilde{a}}$
    $\tilde{\theta}_{p+1} = \tilde{\theta}_p + \alpha_p(\tilde{r}_p + \gamma\tilde{\theta}_p^T\tilde{\phi}_{p+1} - \tilde{\theta}_p^T\tilde{\phi}_p)\tilde{\phi}_p$
  end
  Set $\theta_{t+1} = \tilde{\theta}_{\tau+1}$
end for

Algorithm 2: Gradient-descent and Least-Squares Dyna control algorithms.

where e is the episode index. In the reported figures, the RMSEs of each algorithm were averaged over 30 data sets, each data set comprising of 1000 pre-collected trajectories following the policy. At each planning step, all linear Dyna algorithms sampled unit basis vector, where the "1" component is at a random entry.

The comparisons of TD, gradient descent linear Dyna (gradient Dyna), and LS-Dyna are summarized in Figure 1. Figure 1 (left) shows that, for all values of α, LS-Dyna are continuously better than TD. Figure 1 (right) shows that, for all values of α, LS-Dyna is significantly and consistently better than gradient descent linear Dyna using any β. All linear Dyna algorithms and TD used no eligibility trace. Other parameters were $\theta_0 = 0$, $F_0 = 0$, $f_0 = 0$ and $D_0^{-1} = 100I$.

5.2 Mountain-car

We used the same Mountain-Car environment and tile coding code as in the linear Dyna paper (Sutton et al., 2008). In particular, we used 10 tilings over the continuous, four-dimensional state space. The width of each tile in each dimension is 1/8 the length of the

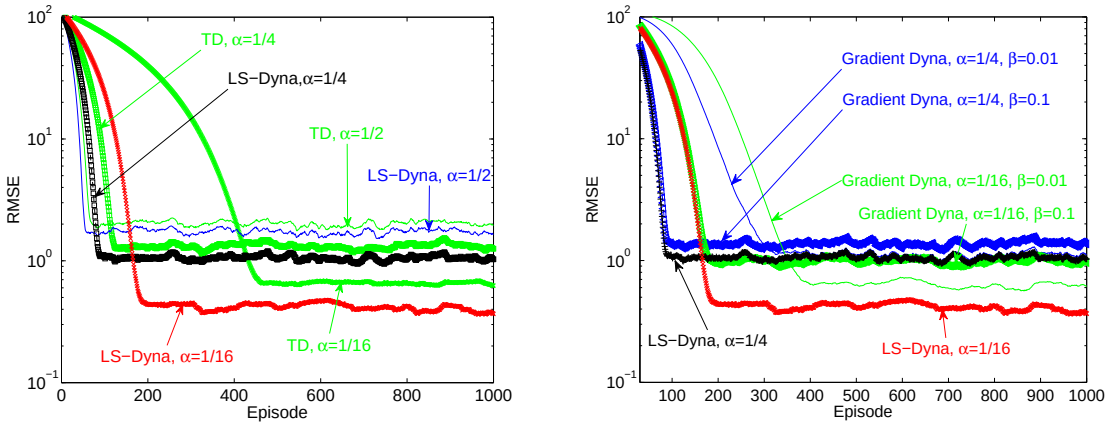

Figure 1: The 98-state Boyan chain example: LS-Dyna are significantly better than linear TD for all values of learning step-sizes (left). Given the same learning step-size, LS-Dyna are much faster than gradient descent linear Dyna (right). All linear Dyna algorithms plan only once and planning step size is 0.1.

dimension. The memory size of the tile is $1,000$. For details of tile coding, please see the RL textbook (Sutton and Barto, 1998). Discounting factor γ is 1.0. No eligibility trace or exploration is used for all algorithms. Each run is composed of 30 episodes. In each episode the algorithms are allowed to try a maximum of 1000 steps. Reward is constantly -1.0 every time step whether the goal is reached or not. Initial weights of all algorithms were drawn from the normal distribution. The effect of this initialization is *optimistic*, in that the initialized value functions are much larger than the true values, and thus it encourages exploration. All reported data was averaged over 50 runs.

Figure 2 (left) shows the trajectory performance of gradient descent linear Dyna, LS-Dyna and Q-learning. Gradient-descent linear Dyna spends much fewer steps in reaching the goal than Q-learning with $\beta = 0.01$ and $\beta = 0.001$. LS-Dyna is the fastest algorithm to achieve the goal of all the compared algorithms. Linear gradient descent Dyna with $\beta = 0.01$ can perform almost as well as LS-Dyna. However, it spends quite a lot of effort in finding the optimal β for gradient descent linear Dyna. On the other hand, LS-Dyna does not have a modeling step-size to tune.

Figure 2 (right) summarizes the steps to goal of LS-Dyna, and gradient descent linear Dyna using various β at the 30th episode. LS-Dyna achieves the best of gradient descent linear Dyna algorithms. The step-size for Q-learning in all Dyna was $\alpha = 0.1$, and the planning step-size in all Dyna was $\alpha_p = 0.1$. Other parameters were $D_0^a = 500I$ (used for LS-Dyna), $F_0^a = I, f_0^a = 0$ (used for LS-Dyna and gradient descent linear Dyna). In planing, all linear Dyna algorithms sampled a random basis unit vector.

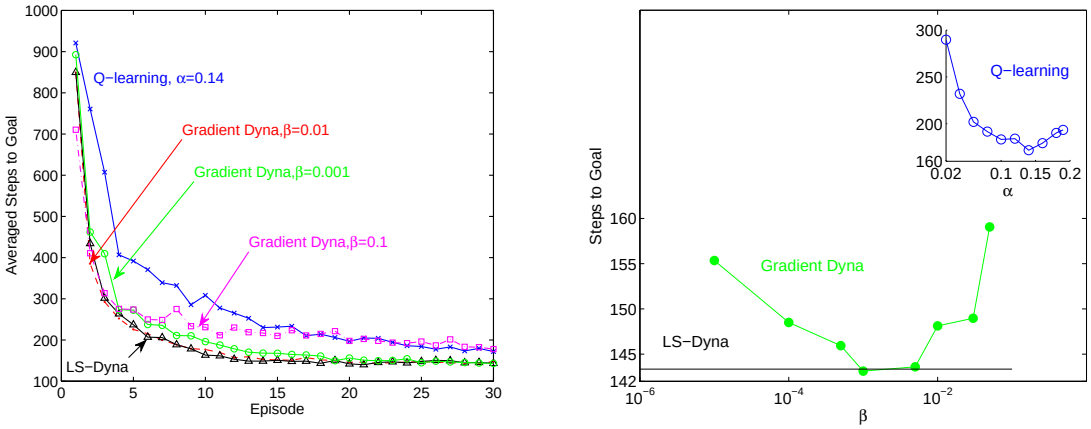

Figure 2: The Mountain-car problem. Left: Trajectory performances of LS-Dyna, linear Dyna and Q-learning. Right: LS-Dyna achieves the best of gradient descent linear Dyna algorithms and Q-learning. All linear Dyna algorithms plan only once.

6. Conclusion

In this paper we proposed LS-Dyna algorithm for planning. The general motivation of LS-Dyna is that the performance of model-based RL algorithms can be improved by a better modeling of the world. LS-Dyna uses recursive least-squares methods to estimate the world model. The advantage of recursive least-squares is that it is more data efficient and free of tuning a modeling step-size. In addition, our LS-Dyna has the same complexity as the gradient descent linear Dyna. We are aware that there has been quite an effort in using least-squares methods to improve the data efficiency of RL controllers in literature. However, previous results show that least-squares versions of TD/Q-learning algorithms for control seems unstable, possibly because the policy changes from time step to step, e.g., see (Sutton et al., 2008; Yao et al., 2009). In our Mountain-car experiment, LS-Dyna was stable and gave much better performance than the gradient descent linear Dyna and Q-learning. The success of LS-Dyna in our experiments may lie in that the world is modeled as the effects of a set of actions, which is usually stationary. Our work implies that action models are useful in representing the dynamics of the world and easier to estimate because of their stationarity. Such features makes action models suitable for planning and improving data efficiency of RL controllers using least-squares methods.

7. Acknowledgement

This paper is based on a course project for "Introduction to Reinforcement Learning" given by Dr. Rich Sutton in Feb 2009. We thank Rich Sutton, Csaba Szepesvari, Amir Massoud, and Istvan Szita for helpful comments.

References

D. P. Bertsekas, V. Borkar, and A. Nedic. Improved temporal difference methods with linear function approximation. In *Learning and Approximate Dynamic Programming*, pages 231–255. IEEE Press, 2004.

J. A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49:233–246, 2002.

S. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

A. Geramifard, M. Bowling, and R. S. Sutton. Incremental least-squares temporal difference learning. In *AAAI*, 2006.

A. Geramifard, M. Bowling, M. Zinkevich, and R. S. Sutton. iLSTD: Eligibility traces and convergence analysis. In *NIPS*, 2007.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

R. S. Sutton, Cs. Szepesvári, A. Geramifard, and M. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *UAI*, 2008.

Richard S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *ICML*, 1990.

X. Xu, H. He, and D. Hu. Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research*, 16:259–292, 2002.

H. Yao and Zhi-Qiang Liu. Preconditioned temporal difference learning. In *ICML*, 2008.

H. Yao, S. Bhatnagar, and D. Diao. Multi-step linear dyna-style planning. In *NIPS*, 2009.