Dyna(k): A Multi-Step Dyna Planning

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

Shalabh Bhatnagar SHALABH@CSA.IISC.ERNET.IN
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India 560 012

Dongcui Diao CREECE.DIAO@GMAIL.COM
School of Economics and Management, South China Normal University, Guangzhou China 518055

Csaba Szepesvári SZEPESVA@CS.UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton, AB Canada T6G2E8

Abstract

Dyna planning is an efficient way of learning from real and imaginary experience. Existing tabular and linear Dyna algorithms are single-step, because an "imaginary" feature is predicted only one step into the future. In this paper, we introduce a multi-step Dyna planning that predicts more steps into the future. Multi-step Dyna is able to figure out a sequence of multi-step results when a real instance happens, given that the instance itself, or a similar experience has been imagined (i.e., simulated from the model) and planned. Our multi-step Dyna is based on a multi-step model, which we call the λ-*model*. The λ-model interpolates between the one-step model and an infinite-step model, and can be learned efficiently online. The multi-step Dyna algorithm, Dyna(k), uses the λ-model to generate predictions k steps ahead of the imagined feature, and applies TD on this imaginary multi-step transitioning.

1. Introduction

Dyna combines knowledge from planning on imaginary experience with that from learning over the real instances. A Dyna agent can be thought of as mimicking human learning by imagining situations that happened much less or did not happen at all. The key idea is to maintain a world model from real interactions with the environment, and apply the world model to generate (i.e., simulate) virtual experience. Planning and learning in Dyna interleave and reinforce each other: at each time step, planning starts from the learned parameters; and the improved parameters after planning are passed back for learning. Planning helps learning in that it provides better parameters for learning and decision making. In turn, we get improved experience that helps refine the world model and thereby also improve planning. In Dyna, we can use various learning algorithms for learning and planning. The classical Dyna uses Temporal Difference (TD) methods (Sutton, 1988; Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998).

Existing Dyna planning algorithms are single-step (Sutton, 1990; Sutton & Barto, 1998; Sutton et al., 2008), because they only simulate one step ahead. This is many times insufficient as one does not exploit in such a case all possible future results. In this paper, we give Dyna planning the multi-step prediction power by using a multi-step model of the world. We first extend the tabular multi-step model of (Sutton, 1995) to linear function approximation, and based on the extension we propose the λ-model of Dyna, which interpolates between the one-step model and the infinite-step model and can be learned efficiently on line. Our multi-step Dyna algorithm, Dyna(k), uses the λ-model to generate k steps ahead prediction of the imagined experience and applies TD learning on it. One extreme of Dyna(k), Dyna(∞), is very efficient in computation as each planning step requires only $O(n)$ computation,

Appearing in Proceedings of the ICML/UAI/COLT Workshop on Abstraction in Reinforcement Learning, Montreal,Canada, 2009. Copyright 2009 by the author(s)/owner(s).

where n is the number of feature functions. The other extreme of Dyna(k), Dyna(1), corresponds to a single-step Dyna that is similar to the existing linear Dyna (Sutton et al., 2008). We compare Dyna algorithms for policy evaluation in the setting of Boyan chain, and the results show that multi-step Dyna is much faster than regular single-step Dyna.

2. TD(0) and Dyna

2.1. TD(0) Algorithm

Given a state space $\mathbb{S} = \{1, 2, \ldots, N\}$, the problem of *policy evaluation* is to predict the long-term reward of a policy π for every state $s \in \mathbb{S}$:

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

where $r(s_t, s_{t+1})$ is the reward received by the agent at time t. Given n ($n \le N$) feature functions $\varphi_j(\cdot) : \mathbb{S} \mapsto \mathbb{R}$, $j = 1, \ldots, n$, the feature of state i is $\phi(i) = [\varphi_1(i), \varphi_2(i), \ldots, \varphi_n(i)]'$. Now V^π can be approximated by $\hat{V}^\pi = \Phi\theta$, where θ is the weight vector, and Φ is the feature matrix whose entries are $\Phi_{i,j} = \varphi_j(i)$, $j = 1, \ldots, n$; $i = 1, \ldots, N$. At time t, if we observe a resulting state s_{t+1} and a reward r_t, a TD error is realized by

$$\delta_t = r_t + \gamma\theta_t'\phi_{t+1} - \theta_t'\phi_t,$$

where ϕ_t corresponds to $\phi(s_t)$. We use prime '$'$' to denote the transpose operator throughout this paper. According to the TD error and the one-step feature, TD(0) adjusts the weights by

$$\theta_{t+1} = \theta_t + \alpha_t\delta_t\phi_t,$$

where α_t is a positive step-size.

2.2. Linear Dyna

Most of the earlier work on Dyna (Sutton, 1990; Sutton & Barto, 1998) uses a lookup table representation of states. Modern Dyna is more advantageous in the use of linear function approximation (Sutton et al., 2008). We denote the state transition probability matrix of policy π by P^π, whose (i,j)th component is $P^\pi_{i,j} = E_\pi\{s_{t+1} = j | s_t = i\}$; and denote the expected reward vector of policy π by R^π, whose ith component is the expected reward of leaving state i in one step. Linear Dyna uses a compressed transition model of policy π:

$$(F^\pi)' = (\Phi' D^\pi \Phi)^{-1} \cdot \Phi' D^\pi P^\pi \Phi,$$

and

$$f^\pi = (\Phi' D^\pi \Phi)^{-1} \cdot \Phi' D^\pi R^\pi,$$

where D^π is an $N \times N$ matrix whose diagonal entries correspond to the steady state distribution under policy π. Here D^π arises because policy evaluation is done on the trajectories following policy π (i.e., the 'on-policy' case).

Dyna repeats some steps of planning in each of which it proposes a sampled feature $\tilde{\phi}$ (we use $\sim$ to denote an imaginary experience). The next feature is shifted from the original *i.e.*, $\tilde{\phi}^{(1)} = F^\pi\tilde{\phi}$. The rewards leaving feature $\tilde{\phi}$ in one step correspond to $\tilde{r}^{(1)} = \tilde{\phi}' f^\pi$. The imaginary experience is therefore $\tilde{\phi} \rightarrow (\tilde{\phi}^{(1)}, \quad \tilde{r}^{(1)})$. Dyna treats this imaginary experience as if it were a real experience, and applies TD learning over it in the following way:

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

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

Matrix F^π and vector f^π constitute the world model of linear Dyna that are estimated at every time step using a gradient descent recursion:

$$F^\pi_{t+1} = F^\pi_t + \alpha_t(\phi_{t+1} - F^\pi_t\phi_t)\phi_t',$$
$$f^\pi_{t+1} = f^\pi_t + \alpha_t(r_t - f^{\pi\prime}_t\phi_t)\phi_t, \tag{1}$$

respectively, where the features and reward are all from real world experience. Under some assumptions, it can be proved that for policy evaluation, the fixed-point of linear Dyna is the same as that of TD(0) (Sutton et al., 2008), which is the solution to a linear system of equations $A^\pi\theta^* + b^\pi = 0$, where

$$A^\pi = \Phi' D^\pi(\gamma P^\pi - I)\Phi, \quad b^\pi = \Phi' D^\pi R^\pi,$$

with I being the identity matrix.

3. The Multi-step Dyna

The general description of multi-step Dyna is shown in Figure 1. Given a situation, multi-step Dyna figures out the sequences of the results in one step, two steps, *etc*, through many "dreams" (i.e., imaginary or model-based experiences) that are connected together; the input to one (dream) being the output from the previous. Predicting is similar to dreaming, however, the input is no longer a proposal but a real instance. When a real instance occurs, multi-step Dyna is able to predict a sequence of multi-step results given that the instance itself or a similar experience has been imagined and planned.

In Section 4 we propose a multi-step model with linear function approximation, the λ-model, which is very efficient in estimating online; and in Section 5 we present the Dyna(k) planning algorithm based on the λ-model.

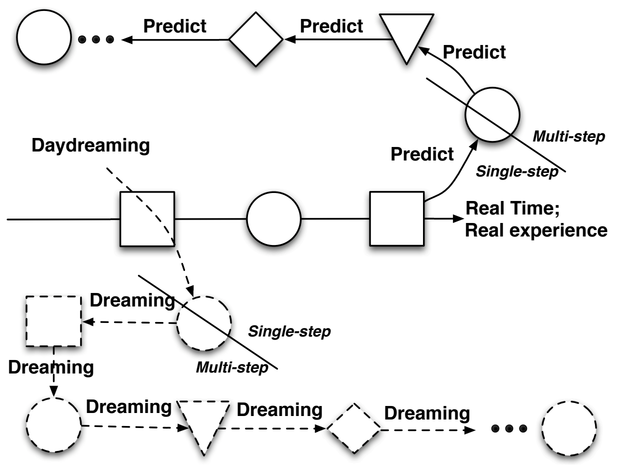

Figure 1. The single and multi-step Dyna. The single-step Dyna dreams only one step ahead and is able to predict only one step into the future, while the multi-step Dyna dreams many steps ahead and is able to predict many steps into the future.

4. The Linear Multi-step Model

In the lookup table representation, the k-step model of the world is given by (Sutton, 1995)

$$P^{(k)} = (\gamma(P^\pi)')^k, R^{(k)} = \sum_{j=0}^{k-1} (\gamma P^\pi)^j R^\pi, k = 1, 2, \dots$$

The k-step model predicts k steps into the future: $P^{(k)}\phi_t$ is the representation of the expected states, k steps in the future and $(R^{(k)})'\phi_t$ is the expected reward in k steps. Notice that

$$V^\pi = R^{(k)} + (P^{(k)})'V^\pi, \quad \forall k = 1, 2, \dots, \qquad (2)$$

which is a generalization of the original Bellman equation $V^\pi = R^\pi + \gamma P^\pi V^\pi$.

4.1. The Optimality of the One-step Linear Model

We show that $(F^\pi)'$ and f^π constitute the best one-step linear model in $\mathbb{R}^{n\times n}$. If the state space is large, it is not practical to maintain the one step model P^π. Instead we maintain a smaller matrix $X \in \mathbb{R}^{n\times n}$, which is a compressed transition matrix in the feature space. For a vector $\theta \in \mathbb{R}^n$, $X\theta$ is its approximated one-step expectation in $\mathbb{R}^n$, while ΦX is an approximation of the one-step transition matrix, P^π.

Given θ and Φ, there are two ways to get the one-step result in the original state space $\mathbb{R}^N$. The first way is to obtain the one-step result in the feature space $\theta \in \mathbb{R}^n$ by applying X, which gives $X\theta$. Then map $X\theta$ back to the state space $\mathbb{R}^N$ to obtain $\Phi X\theta$. Another way is to first map θ back to the state space $\mathbb{R}^N$ and get $\Phi\theta$. Then apply the true model P^π, and get the one-step result, $P^\pi\Phi\theta$. If we want a good X, the error between the two results can be used as a criterion. The following theorem shows that minimizing the error between $\Phi X\theta$ and $P^\pi\Phi\theta$ gives the optimal solution, $X^* = (F^\pi)'$.

Theorem 1. *For $\forall\theta \in \mathbb{R}^n$, the vector $(F^\pi)'\theta$ is the best one-step result of θ in $\mathbb{R}^n$, in the sense that $\Phi((F^\pi)'\theta)$ has the minimum error of approximating the one-step result of any vector $\Phi\theta \in \mathbb{R}^N$*, i.e.,

$$(F^\pi)' = \arg\min_{X\in\mathbb{R}^{n\times n}} ||(\Phi X)\theta - P^\pi\Phi\theta||^2_{D^\pi},$$

where $||\cdot||_{D^\pi}$ is the D^π-norm, defined by $||\Phi\theta||_{D^\pi} = \sqrt{(\Phi\theta)'D^\pi\Phi\theta}$.

Similarly, we have

Theorem 2. *The one-step reward model Φf^π gives the best fit of R^π, in the sense that*

$$f^\pi = \arg\min_{f\in\mathbb{R}^n} ||R^\pi - \Phi f||^2_{D^\pi}.$$

4.2. The Iterated Multi-step Model

Since $(F^\pi)'$ and f^π constitute the best one-step linear model in $\mathbb{R}^{n\times n}$, we can iterate them k times to obtain a k-step model, that we call the *iterated* multi-step model:

$$F^{(k)} = (\gamma F^\pi)^k, \quad f^{(k)} = \sum_{j=0}^{k-1} (\gamma(F^\pi)')^j f^\pi.$$

We extend (2) to a general k-step Bellman equation with linear function approximation:

$$\hat{V}^\pi = \Phi\theta = \Phi f^{(k)} + \Phi(F^{(k)})'\theta, \quad \forall k = 1, 2, \dots. \qquad (3)$$

4.3. The λ-model

Estimating the iterated k-step model $F^{(k)}$ and $f^{(k)}$ on line is intractable for finite $k > 1$. One can first estimate F^π and f^π, and then compute $F^{(k)}$ and $f^{(k)}$ through powers of F^π. However, the matrix product is too complex in computation. Thus instead of estimating the model directly we develop an approximation of it.

First let us see how to use the model. Given an imaginary feature $\tilde{\phi}_\tau$, we look k steps ahead by applying the iterated k-step transition model:

$$\tilde{\phi}^{(k)}_\tau = F^{(k)}\tilde{\phi}_\tau.$$

As k grows, $F^{(k)}$ diminishes quickly and thus $\tilde{\phi}_\tau^{(k)}$ goes to 0 quickly. [1] This means the more steps we look into the future, the more ambiguous is our feature (state). It suggests we can use the following approximated k-step model:

$$F^{(k)} \approx (\lambda\gamma)^{k-1}\gamma F^\pi,$$

where $\lambda \in (0,1]$. To guarantee that the optimality (3) still holds, we approximate the iterated k-step reward model by

$$f^{(k)} \approx (I - (\lambda\gamma)^{k-1}\gamma(F^\pi)')(I - \gamma(F^\pi)')^{-1}f^\pi,$$

When $k=1$, we get back the single-step model used by existing linear Dyna; however, as $k \to \infty$, we get an *infinite-step model.* The intermediate k interpolates between the single-step model and infinite-step model. We call this the λ-*model.* In the infinite-step model, the transition model $(\lambda\gamma)^{k-1}\gamma F^\pi$ diminishes, which is consistent with the fact that the iterated transition matrix in k-step transition model also diminishes as k grows. Thus the infinite-step model reduces to a vector, $f^{(\infty)}$.

For finite k, estimating $F^{(k)}$ has the same complexity with estimating the single-step transition model. For $f^{(k)}$, we have

$$\begin{aligned} f^{(k)} &\approx (I - (\lambda\gamma)^{k-1}\gamma(F^\pi)')(I - \gamma(F^\pi)')^{-1}f^\pi, \\ &= f^{(\infty)} - (\lambda\gamma)^{k-1}\gamma(F^\pi)'f^{(\infty)}, \end{aligned} \tag{4}$$

where we define

$$f^{(\infty)} = (I - \gamma(F^\pi)')^{-1}f^\pi.$$

The case of $k=1$ is interesting. Previous (linear) Dyna algorithm (Sutton et al., 2008) takes advantage of the fact that $f^{(1)} = f^\pi$ and uses gradient descent to estimate it from (1). On the other hand, in our Dyna algorithm, we use (4) and estimate all $f^{(k)}$ from the estimate of $f^{(\infty)}$, that is no longer a gradient estimate for $k=1$. To differentiate between the two versions of the single-step Dyna, we call the linear Dyna in (Sutton et al., 2008) the *gradient descent single-step Dyna*, and denote the single-step Dyna in our framework simply as Dyna(1).

5. Planning using the λ-model

In this section, we present the family of planning algorithms, Dyna(k), that use the λ-model. We first develop a planning algorithm for the infinite-step model, and based on it, we then present Dyna(k) planning for any finite k.

[1] This is because γF^π has a spectral radius smaller than one, cf. Lemma 9.2.2 of (Bertsekas et al., 2004).

5.1. Dyna(∞): Planning using the Infinite-step Model

The infinite-step model is preferable in computation because $F^{(\infty)}$ diminishes and the model reduces to $f^{(\infty)}$. It turns out that $f^{(\infty)}$ can be further simplified to allow an efficient online estimation.

$$\begin{aligned} f^{(\infty)} &= (I - \gamma(F^\pi)')^{-1}f^\pi \\ &= (\Phi'D^\pi\Phi - \gamma\Phi'D^\pi P^\pi\Phi)^{-1} \cdot \Phi'D^\pi\Phi f^\pi \\ &= -(A^\pi)^{-1}b^\pi. \end{aligned} \tag{5}$$

A key problem in modeling the infinite-step model, $f^{(\infty)}$, is to estimate A^π and b^π. One can accumulate these quantities online like LSTD (Bradtke & Barto, 1996; Boyan, 1999). LSTD is data efficient for policy evaluation. However, it is poor for control as all experience is weighted equally (Sutton et al., 2008). In the literature, usually more weight is put on the recent experience. Here we estimate A^π by adopting the Robbins-Monro procedure

$$A_{t+1}^\pi = A_t^\pi + \beta_t(\phi_t(\gamma\phi_{t+1} - \phi_t)' - A_t^\pi). \tag{6}$$

Here β_t, $t \geq 0$ is some positive step-size sequence. When $\beta_t = 1/T$ is used, where T is the number of state visits over all trajectories, A_{t+1}^π is equal to the normalized LSTD matrix. The other interesting extreme is when $\beta_t = 1$, in which case we have $A_{t+1}^\pi = \phi_t(\gamma\phi_{t+1} - \phi_t)'$, which is a rank one matrix used by TD(0). This suggests that step-size can control the amount of experience we can use. In particular, a step-size $\beta_t \in (1/T, 1)$ enables us not only to use all the experience, but also to put larger weights on more recent experiences. However, we do not have to estimate A_{t+1}^π itself, but only $(A_{t+1}^\pi)^{-1}$, which can be updated in $O(n^2)$ using the Sherman-Morrison formula.

$$(A_{t+1}^\pi)^{-1} = \frac{1}{1-\beta_t}\left((A_t^\pi)^{-1} - \frac{\beta_t(A_t^\pi)^{-1}\phi_t d_t'(A_t^\pi)^{-1}}{1 - \beta_t + \beta_t d_t'(A_t^\pi)^{-1}\phi_t}\right), \tag{7}$$

where $d_t = \gamma\phi_{t+1} - \phi_t$. Similarly vector b_t^π can also be estimated by the recency update. Finally $f^{(\infty)}$ can be estimated by $f^{(\infty)} = (A_{t+1}^\pi)^{-1}b_{t+1}^\pi$, where the matrix inversion is computed by (7).

As with traditional Dyna, we initially sample a feature $\tilde{\phi}$ from some distribution μ. We then apply the infinite-step model to get the expected future features and all the possible future rewards:

$$\tilde{\phi}^\infty = F^{(\infty)}\tilde{\phi}, \quad \tilde{r}^\infty = (f^{(\infty)})'\tilde{\phi}.$$

Next, TD(0) is applied on this simulated ("imaginary") experience.

$$\tilde{\theta} := \tilde{\theta} + \alpha(\tilde{r}^\infty + \tilde{\theta}'\tilde{\phi}^\infty - \tilde{\theta}'\tilde{\phi})\tilde{\phi},$$

Algorithm 1 Dyna(k) ($k = 1, \dots, \infty$) planning using the λ-model for evaluating policy π.

```
Initialize $F_0^\pi$, $(A_0^\pi)^{-1}$, $b_0^\pi$ and $\theta_0$
Select an initial state
for each time step do
  Act $a_t$ according to $\pi$ and receive $r_t$, $\phi_{t+1}$
  $d_t = \gamma\phi_{t+1} - \phi_t$
  $\theta_{t+1} = \theta_t + \alpha_t(r_t + d_t'\theta_t)\phi_t$
  $b_{t+1}^\pi = b_t^\pi + \beta_t(\phi_t r_t - b_t^\pi)$
  Compute $(A_{t+1}^\pi)^{-1}$ according to (7)
  $f^{(\infty)} = -(A_{t+1}^\pi)^{-1} b_{t+1}^\pi$
  $F_{t+1}^\pi = F_t^\pi + \alpha_t(\phi_{t+1} - F_t^\pi\phi_t)\phi_t'$,
  $F^{(k)} = (\lambda\gamma)^{k-1}\gamma F_{t+1}^\pi$            /* $F^{(\infty)} = 0$ */
  $f^{(k)} = f^{(\infty)} - (F^{(k)})' f^{(\infty)}$
  Set $\tilde\theta_0 = \theta_{t+1}$
  for $\tau = 1$ to $p$ do
    Sample $\tilde\phi_\tau \sim \mu(\cdot)$
    $\tilde\phi^{(k)} = F^{(k)}\tilde\phi_\tau$            /* $\tilde\phi^{(\infty)} = 0$*/
    $\tilde r^{(k)} = (f^{(k)})'\tilde\phi_\tau$
    $\tilde\theta_{\tau+1} = \tilde\theta_\tau + \alpha_\tau(\tilde r_\tau^{(k)} + \tilde\theta_\tau'\tilde\phi_\tau^{(k)} - \tilde\theta_\tau'\tilde\phi_\tau)\tilde\phi_\tau$
  end for
  Set $\theta_{t+1} = \tilde\theta_{\tau+1}$
end for
```

which simplifies into

$$\tilde\theta := \tilde\theta + \alpha(\tilde r^\infty - \tilde\theta'\tilde\phi)\tilde\phi, \tag{8}$$

because $\tilde\phi^\infty = 0$. The complexity of the infinite-step planning is much lower than the single-step linear Dyna because only $O(n)$ computation is required every planning step here while the single-step linear Dyna requires $O(n^2)$ computation in every planning step.

5.2. Dyna(k): Planning using the λ-model

The k-step λ-model is efficient to estimate, and can be directly derived from the single-step and infinite-step models.

$$F^{(k)} = (\lambda\gamma)^{k-1}\gamma F_{t+1}^\pi,$$

and

$$f^{(k)} = f^{(\infty)} - (F^{(k)})' f^{(\infty)},$$

respectively, where the infinite-step model is estimated by $f^{(\infty)} = (A_{t+1}^\pi)^{-1} b_{t+1}^\pi$. Given an imaginary feature $\tilde\phi$, we look k steps ahead to see the future features and rewards:

$$\tilde\phi^{(k)} = F^{(k)}\tilde\phi, \quad \tilde r^{(k)} = (f^{(k)})'\tilde\phi.$$

Thus we obtain an imaginary k-step transition experience $\tilde\phi \rightarrow (\tilde\phi^{(k)}, \ \tilde r^{(k)})$, on which we can apply an extended form of TD(0):

$$\tilde\theta_{\tau+1} = \tilde\theta_\tau + \alpha(\tilde r^{(k)} + \tilde\theta_\tau'\tilde\phi^{(k)} - \tilde\theta_\tau'\tilde\phi)\tilde\phi$$

We call this algorithm the Dyna(k) planning algorithm, which is shown as Algorithm 1. When $k = 1$, we use a single-step linear Dyna, Dyna(1). Notice that Dyna(1) uses $f^{(\infty)}$ while the gradient descent single-step Dyna uses f^π. When $k \rightarrow \infty$, we obtain the Dyna(∞) algorithm.

6. Boyan Chain Example

The problem we consider is exactly the same as in (Boyan, 1999). The root mean square error (RMSE) for the value function is used as a criterion. We call Dyna(k) for multi-step Dyna that uses the λ-model for all $k = 1, 2, \dots, \infty$. Next, we describe the various parameters of the experiments.

The step-size for TD used was $\alpha = 0.1(1 + 100)/(traj\# + 100)$, which was also the step-size of TD in the learning step of Dyna. For Dyna(k), we used $\beta_t = 1/(T + 1)$ for computing A_t^{-1} and b_t, and and $\alpha_T = 0.5(1 + 10)/(10 + T)$ for F^π computation. For the gradient descent single-step Dyna, F^π and f^π also used the same α_T as above.

All the weights of all the algorithms, f^π for the gradient descent single-step Dyna, and b^π for Dyna(k) were initialized to zero. The LSTD and Dyna(k) matrices were perturbed during initialization. For the gradient descent single-step Dyna, we do not need to perturb the matrix because there is no matrix inversion procedure there. For LSTD, we initialized $A_0 = -0.1I$ while for Dyna(k), $A_0^{-1} = -10I$; and for all Dyna, $F^\pi = 0$.

In the planning step, all Dyna algorithms sampled a feature whose only nonzero component *(1)* is from a uniformly random location. The following experiment only presents the results of planning once, but we found that Dyna is faster when planning more steps.

First we compared the performance of algorithms for a large number of episodes in Figure 2, in which each curve was averaged over 30 (identical) sets of trajectories. Dyna(k) used $\lambda = 0.9$ for $1 < k < \infty$. All Dyna algorithms used $\alpha_\tau = 0.1$. All Dyna algorithms are observed to be significantly faster than TD(0). Furthermore, Dyna algorithms are able to catch up with the accuracy of LSTD. The gradient descent single-step Dyna is a little slower than Dyna(1). Multi-step Dyna algorithms are much faster than the two single-step Dyna algorithms.

Figure 3 shows the performance of Dyna with the planning step-size varied from 0.1 to 1.0. The results suggest that Dyna with a larger planning step-size is much faster than with smaller planning step-size. Across all

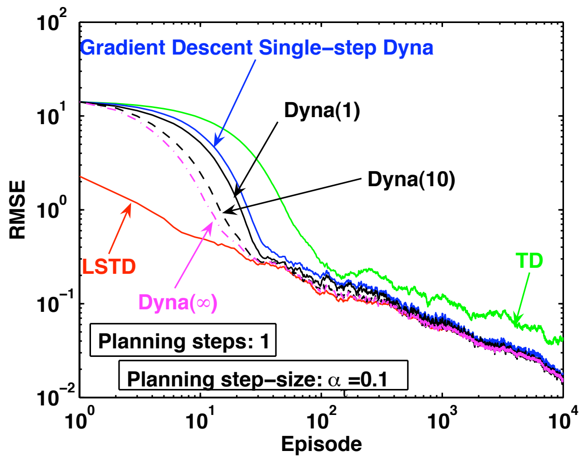

Figure 2. RMSE of TD, the single-step Dyna's, the multi-step Dyna's and LSTD on Boyan chain.

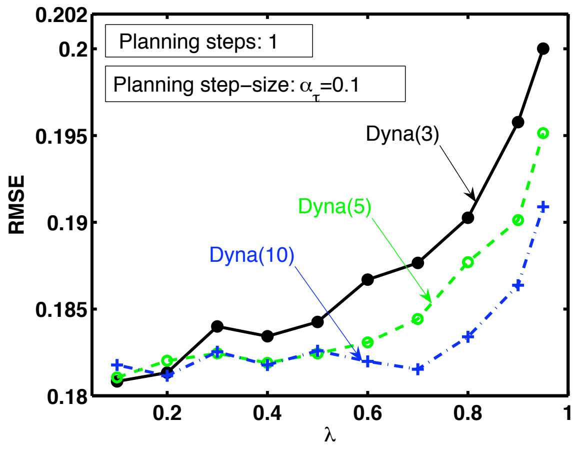

Figure 4. RMSE of Dyna's on Boyan chain for many λ.

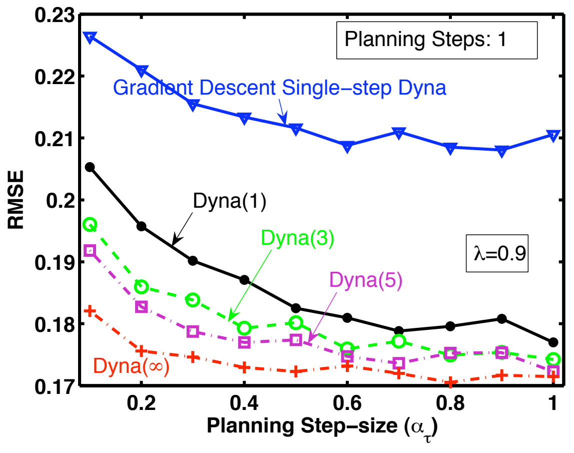

Figure 3. RMSE of Dyna's on Boyan chain for many α.

α_τ, multi-step Dyna is generally significantly faster than single-step Dyna. Figure 4 shows the performance of Dyna with λ varied from 0.1 to 0.95. The results show that smaller λs lead to better performance of Dyna than larger ones. For smaller λ, the RMSE of Dyna are quite close; for bigger λ, multi-step Dyna is significantly faster than single-step Dyna. The curves in Figure 3 and Figure 4 were all averaged over 100 (identical) sets of trajectories.

7. Future Work

We have shown that multi-step Dyna is more advantageous than single-step Dyna for policy evaluation. In addition, we have provided a convergence proof using constant planning step-size that (however) we have not presented here. As part of our future work, we shall extend multi-step Dyna to control and compare the same with the single-step Dyna control algorithm and variants of LSTD.

References

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

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Athena.

Boyan, J. A. (1999). Least-squares temporal difference learning. *ICML-16*.

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

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

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *ICML-17*.

Sutton, R. S. (1995). TD models: Modeling the world at a mixture of time scales. *ICML-12*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Sutton, R. S., Szepesvari, C., Geramifard, A., & Bowling, M. (2008). Dyna-style planning with linear function approximation and prioritized sweeping. *UAI-24*.