Preconditioned Temporal Difference Learning

Hengshuai Yao HENGSHUAI@GMAIL.COM
Zhi-Qiang Liu ZQ.LIU@CITYU.EDU.HK
School of Creative Media, City University of Hong Kong, Hong Kong, China

Abstract

This paper extends many of the recent popular policy evaluation algorithms to a generalized framework that includes least-squares temporal difference (LSTD) learning, least-squares policy evaluation (LSPE) and a variant of incremental LSTD (iLSTD). The basis of this extension is a preconditioning technique that solves a stochastic model equation. This paper also studies three significant issues of the new framework: it presents a new rule of step-size that can be computed online, provides an iterative way to apply preconditioning, and reduces the complexity of related algorithms to near that of temporal difference (TD) learning.

1. Introduction

In Reinforcement Learning (RL), a primary concern is how to reuse experience in an intelligent and fast way. To achieve this we must consider two major issues, namely, the data efficiency and the computational efficiency. Recently these two issues were widely studied by the research on temporal difference (TD) leaning. TD is a classical algorithm well suited for policy evaluation (Sutton, 1988), and achieves great success for its wide applications in control and AI games (Sutton & Barto, 1998). One of its significant advantages is its superior computational efficiency. If the feature vector has K components, TD requires $O(K)$ complexity. However, previous research shows that TD uses experience inefficiently (Lin & Mitchell, 1992)(Geramifard et al., 2006a). The reason is that TD throws the transition information away after using it for one update of weights. One way to reuse this information is to accumulate it into a data set once it has been experienced. Then TD methods are repeatedly applied to the data set. This pattern is known as experience replay (Lin & Mitchell, 1992), and has a high efficiency in using experience because each transition is exploited to the maximum extent. However, this method may be inefficient to perform online because the data set is possible to grow extremely large if the exploration process runs for a long time[1]. Another way is to extract some data structure from the sequence of experience and update the weights with the help of this structure. This way is more desirable because the data structure requires much smaller size of memory than the data set, and leads to recent popular algorithms such as least-squares temporal difference (LSTD) (Boyan, 1999) and least-squares policy evaluation (LSPE) (Nedić & Bertsekas, 2003). Compared to TD, the two algorithms are more data efficient, but similar to the experience replay, are still computationally expensive.

LSTD inverts some accumulated matrix per time step, and generally requires $O(K^3)$. Recursive LSTD (RLSTD) computes the inversion of LSTD's matrix iteratively using Sherman-Morison formula and reduces the complexity to $O(K^2)$ (Bradtke & Barto, 1996)(Xu et al., 2002). LSPE is similar to LSTD and will be examined later. Recently incremental LSTD (iLSTD) was proposed to strike a balance between LSTD and TD (Geramifard et al., 2006a)(Geramifard et al., 2006b): its data efficiency is almost as good as LSTD, but its computational cost is very near to that of TD. However, iLSTD still requires tuning the step-size manually as TD. In contrast, LSTD has no parameter to tune.

The aim of this paper is to explore the relations among recent popular policy evaluation algorithms. A framework of policy evaluation algorithms called the *preconditioned temporal difference* (PTD) learning is introduced, which includes LSTD, LSPE and a variant of iLSTD, *etc.* Furthermore, we maintain LSTD's prop-

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

[1] Lin avoided this problem by using only a window of most recent experience (Lin & Mitchell, 1992). This, however, results in loss of information and it is in general difficult to prespecify the window size.

erty of ease of tuning. This paper proposes an adaptive step-size that can be computed online by all PTD algorithms.

To reduce computational complexity $O(K^2)$ of the PTD algorithms due to the inversion of the preconditioner matrix, we develop an efficient incremental process to apply preconditioning, which leads to a set of incremental PTD algorithms. Incremental PTD algorithms take advantage of the sparse nature of RL tasks, storing and manipulating only the valid experience by a *condensed* structure every time step. We also present an efficient adaptive step-size for incremental PTD algorithms. Results on Boyan chain example show that PTD algorithms using the adaptive step-size gives much faster convergence than using previous rule of step-size; incremental PTD algorithms via condensed implementation have a high efficiency in using data while a low complexity similar to iLSTD.

1.1. Stationary Model Equation

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

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

where γ is the discount factor, and $r(s_t, s_{t+1})$ is the reward received by the agent at time t. Given $K (K \leq N)$ feature functions $\varphi_k(\cdot) : \mathcal{S} \mapsto \mathcal{R}$, $k = 1, \ldots, K$, the feature of state s_t is $\phi_t = [\varphi_1(s_t), \varphi_2(s_t), \ldots, \varphi_K(s_t)]'$. The optimal reward vector J can now be approximated by $\hat{J} = \Phi w$, where w is the weight vector, and Φ is the feature matrix whose entries are $\Phi(j, k) = \varphi_k(j)$, $k = 1, \ldots, K$; $j = 1, \ldots, N$.

For an ergodic Markov chain that has steady-state probabilities $\pi(1), \pi(2), \ldots, \pi(N)$, (Tsitsiklis & Van Roy, 1997) proved that TD(λ) eventually finds a weight vector w^* that satisfies a linear system

$$Aw^* = -b, \tag{1}$$

where A and b are defined by

$$A = \Phi' D(\gamma P - I) \sum_{k=0}^{\infty} (\lambda\gamma P)^k \Phi, \quad b = \Phi' D \sum_{k=0}^{\infty} (\lambda\gamma P)^k \bar{r},$$

D is the diagonal matrix with diagonal entries $\pi(i)$, $i = 1, \ldots, N$; $\lambda \in [0, 1]$ is the eligibility trace factor; P is the transition probability matrix; I is the identity matrix; and $\bar{r}$ is the vector with components $\bar{r}_i = \sum_{j=1}^{N} P_{i,j} r(i, j)$, $i = 1, \ldots, N$. For each $\lambda \in [0, 1]$, w^* is also the limit point of LSTD(λ), LSPE(λ) and iLSTD(λ). Equation (1) is only useful for analysis but not applicable in practice and will be called the stationary model equation.

1.2. Law of Large Numbers

A common structure grown by LSTD(λ), LSPE(λ) and iLSTD(λ) is updated incrementally. If the current transition from s_t to s_{t+1} incurs a reward r_t, then a matrix and a vector are updated by

$$\tilde{A}_{t+1} = \tilde{A}_t + z_t(\gamma\phi_{t+1} - \phi_t)'; \quad \tilde{b}_{t+1} = \tilde{b}_t + z_t r_t,$$

where z_t is the eligibility trace, computed recursively by $z_{t+1} = \lambda\gamma z_t + \phi_{t+1}$. Because the components of $\tilde{A}_{t+1}$ and $\tilde{b}_{t+1}$ can get to infinity it is better to use some well-defined term. For infinite-horizon problems, (Tadić, 2001)(Nedić & Bertsekas, 2003) used the following structure

$$A_{t+1} = \frac{1}{t+1}\tilde{A}_{t+1}; \quad b_{t+1} = \frac{1}{t+1}\tilde{b}_{t+1}, \tag{2}$$

which satisfies the law of large numbers. However, (2) are no longer consistent estimations of A and b for absorbing Markov chains such as Boyan chain example. In Section 2.1, such an extension is proposed.

1.3. Related Algorithms

At time t, the rule of LSTD(λ) for updating weights can be specified as $w_{t+1} = -\tilde{A}_{t+1}^{-1}\tilde{b}_{t+1}$. In practice, $\tilde{A}_t$ can be singular and a perturbation which sets $\tilde{A}_0$ to $\delta_0^- I$ ($\delta_0^- < 0$) should be used.

LSPE(λ) was proposed for infinite-horizon problem (Nedić & Bertsekas, 2003). If the current step-size is α_t, LSPE updates w by

$$w_{t+1} = w_t + \alpha_t (D_{t+1})^{-1} (A_{t+1} w_t + b_{t+1}), \tag{3}$$

where

$$D_{t+1} = \frac{1}{t+1}\left(\delta_0^+ I + \sum_{k=0}^{t} \phi_k \phi_k'\right), \quad \delta_0^+ > 0.$$

In the long run, D_{t+1} converges to $\Phi' D \Phi$.

1.4. Preconditioning

Generally, solutions to a linear system like (1) can be categorized into two classes. The first is the direct methods (Saad, 2003), which factorize A into easily invertible matrices, including Gaussian elimination and LU/QR factorization, *etc.* However, the complexity involved in factorizing A is not practical for large scale systems. The second class, known as the iterative solutions, scales well with the problem size and is very efficient for large and sparse linear systems.

According to the literature of iterative solutions, preconditioning is especially effective for symmetric system (Saad, 2003), but usually for RL tasks, matrix A is not symmetric. Therefore, the original stationary model equation is first transformed into the following symmetric form

$$A'Aw^* = -A'b, \tag{4}$$

which can be solved by Richardson's iteration (Saad, 2003)

$$w_{\tau+1} = w_\tau - \alpha A' \left(Aw_\tau + b\right), \tag{5}$$

where α is some positive scalar that should satisfy

$$\rho(I - \alpha A'A) < 1.$$

The technique of preconditioning refers to a general technique which preconditions a system before solving it. For example, preconditioning (4) gives us the preconditioned symmetric model equation

$$C^{-1}A'Aw^* = -C^{-1}A'b,$$

where C is an invertible matrix, usually called the *preconditioner*. Then the model equation is solved by the iteration

$$w_{\tau+1} = w_\tau - C^{-1}A' \left(Aw_\tau + b\right). \tag{6}$$

Convergence rate of (6) is governed by the spectral radius of $I - C^{-1}A'A$: the smaller the radius is, the faster the solution will be (Saad, 2003). Therefore a good preconditioner should make the preconditioned radius smaller than the original radius, *i.e.*, $\rho(I - C^{-1}A'A) < \rho(I - \alpha A'A)$.

2. The Generalized Framework

We first give consistent estimations of A and b for absorbing Markov chains, and then we show how to apply them together with preconditioning to policy evaluation.

2.1. Robbins-Monro for Absorbing Chains

A trajectory of an absorbing Markov chain is a finite sequence $s_0, \ldots, s_q$, where s_q is the absorbing state. Given trajectories $1, \ldots, M$, where the mth trajectory has length L_m, the consistent estimations of A and b are

$$A_M = \frac{1}{T}\sum_{m=1}^{M}\sum_{t=0}^{L_m} z_t(\gamma\phi_{t+1} - \phi_t)', \tag{7}$$

and

$$b_M = \frac{1}{T}\sum_{m=1}^{M}\sum_{t=0}^{L_m} z_t r_t, \tag{8}$$

where T is the number of all observed state visits in M trajectories, and z_t is the eligibility trace. Similarly, for absorbing Markov chain, LSPE should use the following structure to estimate $\Phi' D \Phi$:

$$D_M = \frac{1}{T}\sum_{m=1}^{M}\sum_{t=0}^{L_m} \phi_t\phi_t'. \tag{9}$$

On a transition from s_t to s_{t+1}, estimations (7), (8) and (9) can be updated incrementally, which is achieved by a Robbins-Monro (RM) procedure:

$$A_{t+1} = A_t + \frac{1}{T}(z_t(\gamma\phi_{t+1} - \phi_t)' - A_t), \tag{10}$$

$$b_{t+1} = b_t + \frac{1}{T}(z_t r_t - b_t), \tag{11}$$

and

$$D_{t+1} = D_t + \frac{1}{T}(\phi_t\phi_t' - D_t), \tag{12}$$

where T is updated by $T = T + 1$ after the three estimations. The convergence of RM procedure can be proved in similar manner to the case of infinite-horizon problems (Tadić, 2001)(Nedić & Bertsekas, 2003).

2.2. The Framework

Given A_{t+1} and b_{t+1} estimated by RM, we can define a *stochastic model equation*

$$A_{t+1}w = -b_{t+1}.$$

Because RM estimations have some error, the stochastic model equation is not satisfied, and there exists a nonzero residual vector

$$e_{t+1}(w) = A_{t+1}w + b_{t+1}. \tag{13}$$

A natural idea is that the current weights can be improved by minimizing the residual error $||e_{t+1}(w)||^2$, which produces a gradient descent algorithm

$$w_{t+1} = w_t - \alpha_t A'_{t+1}(A_{t+1}w_t + b_{t+1}),$$

where α_t is a positive step-size. Gradient descent algorithm is a stochastic form of the iteration (5).

The general preconditioned temporal difference (PTD) learning applies the technique of preconditioning to improve the convergence rate of gradient descent. Assume C_{t+1} is a chosen preconditioner, the rule of PTD can be cast as

$$w_{t+1} = w_t - \alpha_t C_{t+1}^{-1} A'_{t+1}(A_{t+1}w_t + b_{t+1}), \tag{14}$$

where α_t is some scalar but not necessarily positive. With the rule proposed in Section 3, the step-size guarantees the convergence of PTD algorithms and makes

them more flexible in stochastic environments than (6).

The choice of preconditioner is a key issue. Generally, preconditioner should decrease the spectral radius of gradient descent:

$$\rho(I - \alpha_t C_{t+1}^{-1} A'_{t+1} A_{t+1}) < \rho(I - \alpha_t A'_{t+1} A_{t+1}).$$

Gradient descent makes no preconditioning because it chooses the identity matrix as preconditioner. Good examples of preconditioner can be found in recent popular policy evaluation algorithms.

2.3. Relations to Previous Algorithms

LSTD, LSPE and iLSTD are all special forms of applying preconditioner to gradient descent algorithm:

$C_{t+1} = -A'_{t+1} D_{t+1}$, where D_{t+1} is defined in (12). One can easily verify that this is a variant of LSPE(λ).

$C_{t+1} = A'_{t+1} A_{t+1}$. This is an extended form of LSTD: $w_{t+1} = (1 - \alpha_t) w_t + \alpha_t (-A_{t+1}^{-1} b_{t+1})$. Using 1 as the step-size, we get exactly LSTD(λ). Later we will see that LSTD is optimal in choosing its step-size because certain residual error is minimized.

$C_{t+1} = -A'_{t+1}$. This approach is a variant of iLSTD(λ) (Yao & Liu, 2008).

3. The Rule of Step-size

This section presents an adaptive process to compute the step-size online for gradient descent algorithm, LSTD, iLSTD and LSPE. The four algorithms all use a preconditioner in the form of $A'_{t+1} B_{t+1}$, which is assumed to be used by general PTD algorithms.

The update direction of PTD is provided by a *preconditioned residual* (p-residual) vector

$$\delta_t = B_{t+1}^{-1} e_{t+1}(w_t), \tag{15}$$

where e_{t+1} is the residual vector defined in (13). This p-residual is an "old" one, because it is obtained before the weight update. After the weight update, the p-residual vector changes to

$$\theta_{t+1} = B_{t+1}^{-1} e_{t+1}(w_{t+1}).$$

From (14) and (15), θ_{t+1} can be rewritten as

$$\begin{aligned} \theta_{t+1} &= B_{t+1}^{-1}(A_{t+1}(w_t - \alpha_t \delta_t) + b_{t+1}) \\ &= \delta_t - \alpha_t B_{t+1}^{-1} A_{t+1} \delta_t. \end{aligned}$$

Because θ_{t+1} stands for an improved difference between the two sides of the preconditioned stochastic model equation, naturally we hope that the new p-residual error is smaller than the old one: $||\theta_{t+1}||^2 < ||\delta_t||^2$. This can be guaranteed by requiring that θ_{t+1} be orthogonal to $\theta_{t+1} - \delta_t$. Accordingly, we obtain a new rule of step-size

$$\alpha_t = \frac{\delta'_t B_{t+1}^{-1} A_{t+1} \delta_t}{(B_{t+1}^{-1} A_{t+1} \delta_t)'(B_{t+1}^{-1} A_{t+1} \delta_t)}. \tag{16}$$

This step-size is the optimal value that minimizes the new p-residual error over $\alpha \in \mathcal{R}$, *i.e.*, $\alpha_t = \arg\min ||\theta_{t+1}||^2$. Obviously the step-size is positive when $B_{t+1}^{-1} A_{t+1}$ is positive definite, which is true for gradient descent algorithm, iLSTD, and LSTD.

It is interesting that in (16), if $B_{t+1} = A_{t+1}$, then the step-size is equal to 1. This indicates that LSTD's choice of step-size is optimal in the sense that the p-residual error $||(1 - \alpha_t)(w_t + A_{t+1}^{-1} b_{t+1})||^2$ is minimized. When $B_{t+1} = -I$, the residual error $||(I + \alpha_t A_{t+1}) e_{t+1}(w_t)||^2$ is minimized; for ease of later comparisons with previous step-size of iLSTD, this variant will be called the Minimal Residual (MR) algorithm.

To compute p-residual vector and step-size, PTD algorithms have to carry out matrix inversion, which requires $O(K^3)$. Sherman-Morison formula is a solution to reduce this complexity to $O(K^2)$. Another efficient solution is to apply preconditioning incrementally and take advantage of the sparse nature of RL tasks.

4. Incremental PTD

The key of incremental preconditioning is to approximate the p-residual and the adaptive step-size in an iterative way.

4.1. Iterative P-residual and Approximated Step-size

Let κ_t be the error caused by the residual and the current iterative p-residual, defined by

$$\kappa_t = e_{t+1}(w_t) - B_{t+1} \hat{\delta}_t. \tag{17}$$

The new estimation of δ_t can be improved by

$$\hat{\delta}_{t+1} = \hat{\delta}_t - \beta_t \kappa_t, \tag{18}$$

where β_t is computed by

$$\beta_t = -\frac{\kappa'_t (B_{t+1} \kappa_t)}{(B_{t+1} \kappa_t)'(B_{t+1} \kappa_t)}. \tag{19}$$

Substituting the p-residual δ_t in (14) with the iterative p-residual $\hat{\delta}_t$, we get the general form of incremental PTD algorithms

$$w_{t+1} = w_t - \hat{\alpha}_t \hat{\delta}_t, \tag{20}$$

Algorithm 1 Efficient matrix-vector multiplication using CSR.

Input: $Z_Q^{t+1}(a, c, d)$ and a vector β_t
Output: A vector $o_t = Q_{t+1}\beta_t$
for $k = 1$ **to** K **do**
  $k_1 = d_{t+1}(k)$
  $k_2 = d_{t+1}(k+1) - 1$
  $o_t(k) = a_{t+1}(k_1 : k_2)'\beta_t(c_{t+1}(k_1 : k_2))$
end for

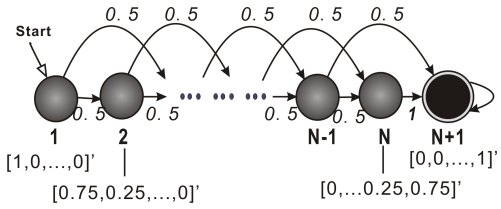

Figure 1. Boyan chain example with $N + 1$ states. The transition probabilities are marked on the arch.

where $\hat{\alpha}_t$ is computed by the following steps.

Given the iterative p-residual $\hat{\delta}_t$, steps (21a)–(21d) compute a vector v, which is an approximation of $B_{t+1}^{-1}A_{t+1}\delta_t$; then the approximated step-size is computed by (21e):

$$\xi_t = A_{t+1}\hat{\delta}_t, \tag{21a}$$

$$\chi_t = \xi_t - B_{t+1}v_t, \tag{21b}$$

$$\eta_t = -\frac{\chi_t'(B_{t+1}\chi_t)}{(B_{t+1}\chi_t)'(B_{t+1}\chi_t)}, \tag{21c}$$

$$v_{t+1} = v_t - \eta_t\chi_t, \tag{21d}$$

$$\hat{\alpha}_t = \frac{\hat{\delta}_t'v_{t+1}}{v_{t+1}'v_{t+1}}. \tag{21e}$$

If $B_{t+1} = -I$, iterative p-residual reproduces p-residual exactly and we get MR(λ); If $B_{t+1} = -D_{t+1}$, we get an incremental form of LSPE(λ) (iLSPE(λ)) that applies preconditioning via iterative p-residual.

4.2. Incremental PTD Using CSR

If the function approximation used is sparse, then matrix Φ is sparse. While this seems a restrictive condition, several popular linear function approximation schemes such as lookup table, Boyan's linear interpolation approximation and tile coding (Sutton & Barto, 1998), are indeed sparse. If the transition matrix is also sparse, matrices A_{t+1} and B_{t+1} will both have many zero entries, implying that "no experience is available for the states related to these entries". Therefore, it is better to remove the void experience and store only the valid experience by a *condensed* structure. Here the Compressed Sparse Row (CSR) format (Saad, 2003) is used. Let Q_t stand for A_t or B_t. The CSR format is a triplet $Z_Q^t(a_t, c_t, d_t)$, where a_t is a real array containing all the real values of the nonzero elements of Q_t; c_t is an integer array containing the column indices of the elements stored in a_t; and d_t is an integer array containing the pointers to the beginning of each row in a_t and c_t.

When Q_{t+1} is sparse, the need for fast matrix-vector multiplication offers a place where CSR fits in. The details are shown by Algorithm 1, whose complexity is $O(l_{t+1})$, where l_{t+1} is the number of nonzero entries in Q_{t+1}. Now (17), (19), (21a), (21b) and (21c) can make a call to Algorithm 1, and the complexity of incremental PTD is given by the following theorem which is proved in (Yao & Liu, 2008).

Theorem 4.2.1 (Complexity of incremental PTD)**.** *The per-time-step complexity of incremental PTD using CSR is $O(qK)$, where q is a small positive real related to the sparsity of matrix A.*

5. Boyan Chain Example

Boyan chain and the features are shown in Figure 1. Transition from N to $N + 1$ incurs a reward -2; transition from $N + 1$ incurs 0; the other transitions incur -3. The discount factor γ is set to 1.

The first experiment is another implementation of experience replay. As RM procedure is able to extract compressed experience information by estimations of A and b, it is natural to ask whether experience can be well replayed by repeatedly presenting RM's estimations to PTD algorithms. Two questions arise for this approach. Will it lead to convergence? What is the role of λ for PTD(λ)?

RM(λ) were first run and averaged over 10 sets of 10000 trajectories, and then their estimated structures were repeatedly presented to Gradient descent(GRAD(λ)), iLSTD(λ), LSPE(λ) and LSTD(λ). All compared algorithms used the adaptive step-size derived in Section 3, and converged to satisfactory solutions for a variety of problem sizes with all $\lambda \in [0, 1]$. The case of $N = 12$ is shown in Figure 2. It is very interesting that for all PTD algorithms smaller λ gives better performance; $\lambda = 0$ performs best and $\lambda = 1$ performs worst, —exactly the same role with that for TD(λ) under repeated presentation training paradigm (Sutton, 1988). Explanation can be given if we view TD as a model exploration process: although TD does not use the model A and b explicitly, its learn-

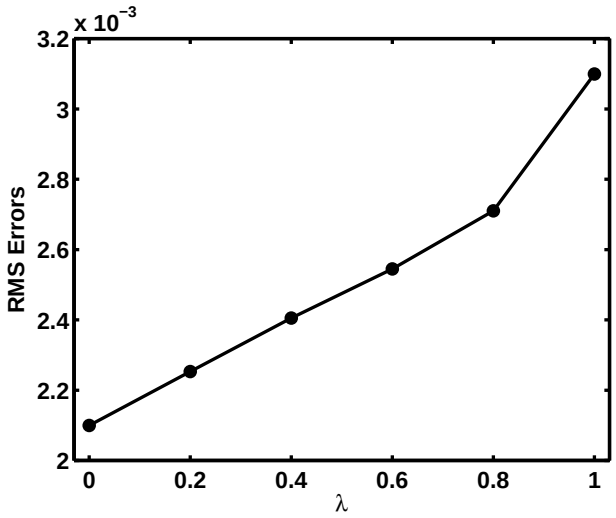

Figure 2. Effects of λ: the (same) RMS errors by GRAD(λ), MR(λ), LSPE(λ) and LSTD(λ).

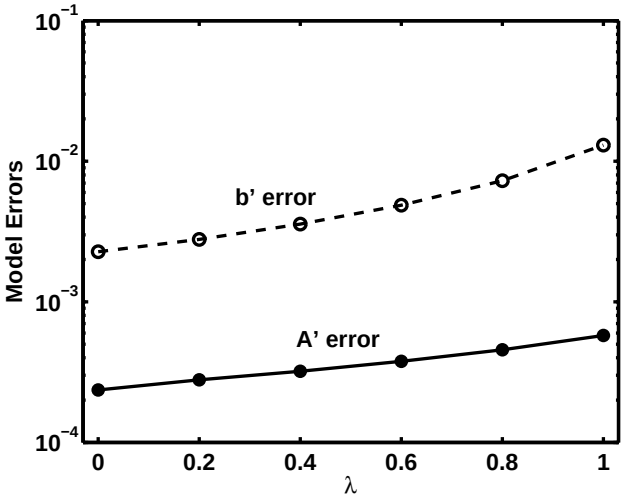

Figure 3. Model errors by RM(λ) procedures.

ing requires exploring and sampling temporal values of the model. It appears that both TD and PTD algorithms rely on the model data reflected by the sets of trajectories.

To explore λ's effect for algorithms, we only have to study its role for the model data, which is extracted by RM procedure. Results are shown in Figure 3, where the model errors are measured by $||A_T - A||_2$ and $||b_T - b||_2$, averaged over 10 sets of 10000 trajectories. It can be observed that smaller λ has smaller modeling errors for A and b, —the role of λ for RM(λ) is just what should be consistent with that for TD(λ) and PTD(λ).

Although all PTD algorithms converge to the same solution, their rates of convergence are quite different. The case of $\lambda = 1$ is shown in Figure 4. MR, LSPE and LSTD are faster than Gradient descent because they make use of preconditioning and their spectral radii are smaller than that of Gradient descent. Figure 5 compares the spectral radii $\rho(I - \alpha_t C_t^{-1} A_t' A_t)$ of different algorithms.

Algorithms were also compared under the same learning paradigm as Boyan (Boyan, 1999), where weights were updated immediately after RM estimations at each time step. All compared algorithms used the adaptive rule except that iLSPE used the approximate step-size developed in Section 4.1. For both adaptive step-size and approximated step-size, a satisfactory convergence was obtained. Results are shown in Figure 6 and Figure 7, where each point was the averaged RMS error over 10 sets of data. It is clear that some intermediate value of λ performs best in both learning error and convergence rate for all algorithms. Generally, four preconditioned algorithms learns faster than Gradient descent algorithm. However, the convergence rate advantages of MR(λ), iLSPE(λ), LSPE(λ) and LSTD(λ) over Gradient descent are becoming smaller as λ increases. The reason may be that larger λ causes larger model error and deteriorates the effects of preconditioning.

Experiment was also run to compare the adaptive step-size with the rule used by (Geramifard et al., 2006a), which takes $\alpha_t = \frac{c_0(c_1+1)}{traj\#+c_1}$, where c_0 was chosen from $\{0.01, 0.1, 1\}$, and c_1 was chosen from $\{100, 1000, 10^6\}$. The best performance of all the nine combinations of the two constants was experimentally chosen for iLSTD. Figure 8 shows that RMS error of MR (adaptive step-size) is faster to decrease than that of iLSTD. From Figure 8, we can also observe that PTD's predictions (such as those given by LSTD and LSPE) have larger variations than incremental PTD's (such as those given by MR and iLSPE). The reason is that PTD algorithms are based on the inversion of preconditioner, which is not well conditioned at the beginning stage of learning; while incremental PTD algorithms avoid numerical instability via iterative p-residual.

Table 1 compares the complexity of PTD and incremental PTD algorithms, where CSR are used for MR (one CSR for A_t) and iLSPE (one CSR for A_t and one CSR for D_t). We can see that incremental PTD algorithms have a clear computational advantage over

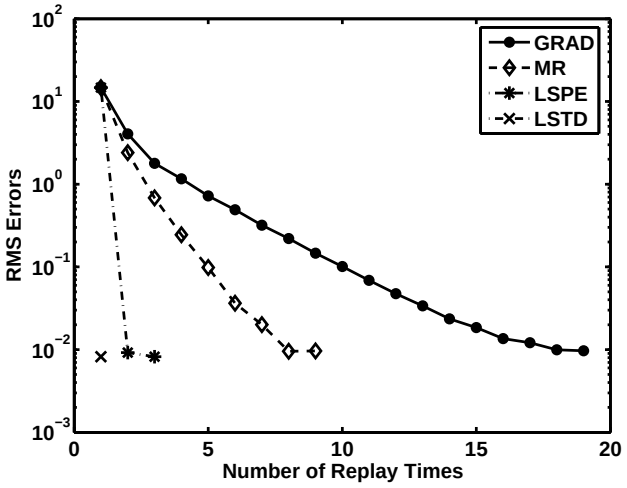

Figure 4. The role of preconditioner ($\lambda = 1$). Algorithms are stopped if RMS error is smaller than 0.01.

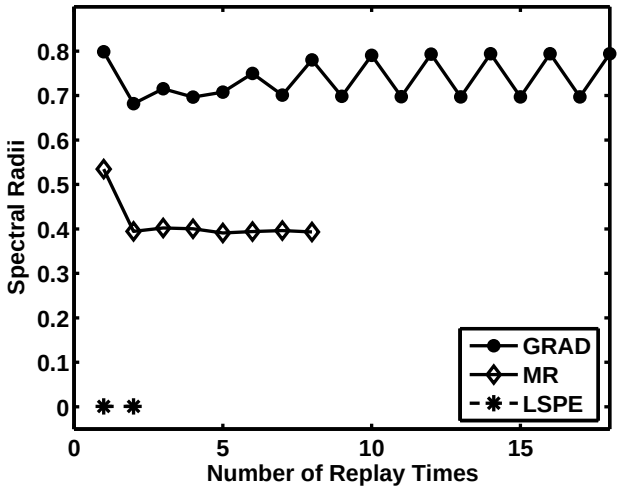

Figure 5. Spectral radius comparisons ($\lambda = 1$). LSTD's spectral radius is 0 permanently, thus not shown.

Table 1. Comparison of per-time-step running time (ms) of PTD and incremental PTD algorithms on $K = 401$ ($\lambda = 0$). The machine used is Pentium(R) 4 PC (CPU 3.00GHZ; RAM 1.00GB).

LSTD	LSPE	iLSTD	MR	iLSPE
72.3	120.3	5.9	10.6	21.9

Table 2. Comparison of memory requirements for A using CSR and full matrix for a variety of problem sizes ($\lambda = 0$).

N	12	100	400	800	1200	1600
$\frac{l}{K^2}$	0.75	0.1479	0.04	0.0198	0.0132	0.01

PTD algorithms. Reason lies in that CSR enables incremental PTD to manipulate much smaller size of data than PTD. Table 2 shows the relative memory requirements of CSR and full matrix for a variety of problem sizes by the ratio l/K^2, where l is the nonzero entries of A. We can observe that the larger the size of state space is, the more advantages will be gained by using CSR.

6. Conclusion

In this paper we proposed two general frameworks, PTD and incremental PTD, which are more data efficient than TD. Generally PTD approaches such as LSTD and LSPE are computationally expensive, whereas incremental PTD algorithms such as MR and iLSPE can take advantage of sparse nature of RL tasks, and have complexity near to that of iLSTD and TD. We also develop an adaptive process for computing the step-size online for PTD algorithms, and an approximated process for computing the step-size for incremental PTD algorithms.

Acknowledgement

We are thankful to Lihong Li, George Konidaris and Andrew Barto for helpful discussions with a draft of this paper. We appreciate for the suggestions by the four reviewers that help improve this paper in many aspects. This research has been partially supported by RGC CERG grant No. CityU 1178/06 (9041147) from Hong Kong UGC.

References

Boyan, J. A. (1999). Least-squares temporal difference learning. *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 49–56). Morgan Kaufmann.

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

Geramifard, A., Bowling, M., & Sutton, R. S. (2006a). Incremental least-squares temporal difference learning. *Twenty-First National Conference on Artificial Intelligence (AAAI-06)* (pp. 356–361). AAAI Press.

Geramifard, A., Bowling, M., Zinkevich, M., & Sutton, R. S. (2006b). iLSTD: Eligibility traces and

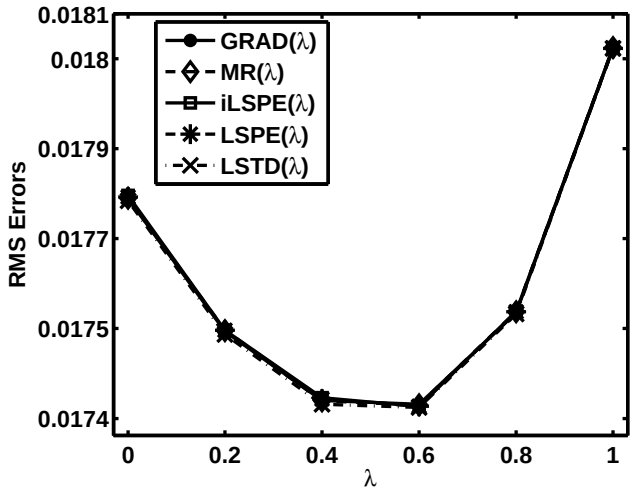

Figure 6. The RMS errors at 90000th visit by GRAD(λ), MR(λ), iLSPE(λ), LSPE(λ) and LSTD(λ).

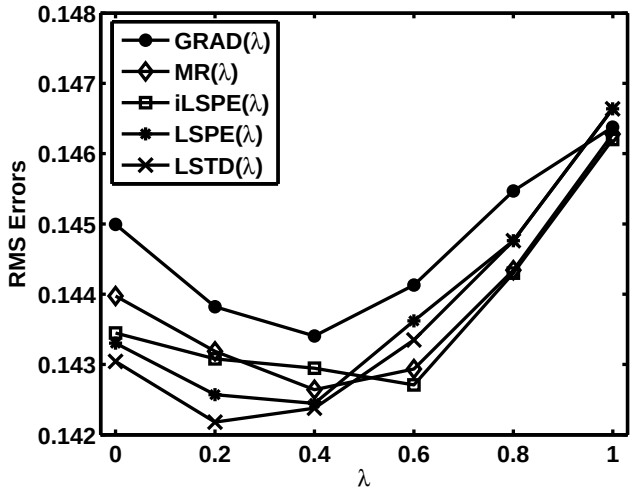

Figure 7. Comparison of convergence rate in terms of RMS errors at 900th state visit.

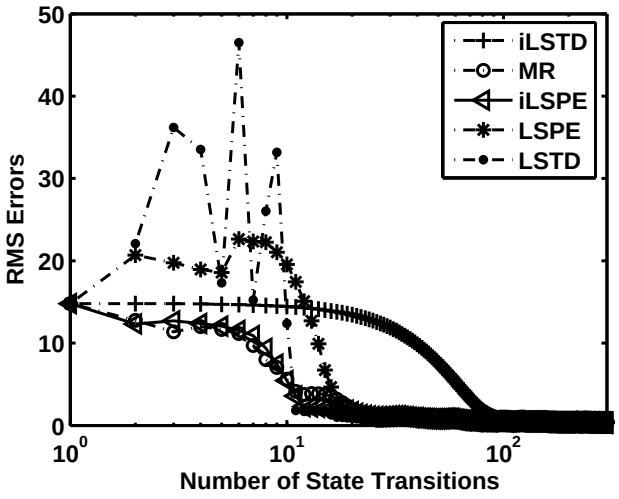

Figure 8. Averaged RMS errors (over 10×10000 trajectories) of iLSTD, MR, iLSPE, LSPE and LSTD in Boyan's setting ($\lambda = 0$). Perturbation factors of LSTD and LSPE were -0.01 and 0.01 respectively. Weights for all algorithms were initialized to $[0.1, 0.1, 0.1, 0.1]'$ except for LSTD.

convergence analysis. *Advances in Neural Information Processing Systems 19* (pp. 441–448).

Lin, L.-J., & Mitchell, T. M. (1992). *Memory approaches to reinforcement learning in non-markovian domains* (Technical Report CMU-CS-92-138). Carnegie Mellon University, Pittsburgh, PA 15213.

Nedić, A., & Bertsekas, D. P. (2003). Least-squares policy evaluation algorithms with linear function approximation. *Journal of Discrete Event Systems*, *13*, 79–110.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.

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

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

Tadić, V. (2001). On the convergence of temporal-difference learning with linear function approximation. *Machine Learning*, *42*, 241–267.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

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

Yao, H., & Liu, Z. (2008). *Preconditioned temporal difference learning* (Technical Report CityU-SCM-MCG-0408). City University of Hong Kong.