Tracking Value Function Dynamics to Improve Reinforcement Learning with Piecewise Linear Function Approximation

Chee Wee Phua CHEEWEE.PHUA@NICTA.COM.AU
Robert Fitch ROBERT.FITCH@NICTA.COM.AU
NICTA, University of New South Wales, Sydney NSW 2052, Australia

Abstract

Reinforcement learning algorithms can become unstable when combined with linear function approximation. Algorithms that minimize the mean-square Bellman error are guaranteed to converge, but often do so slowly or are computationally expensive. In this paper, we propose to improve the convergence speed of piecewise linear function approximation by tracking the dynamics of the value function with the Kalman filter using a random-walk model. We cast this as a general framework in which we implement the TD, Q-Learning and MAXQ algorithms for different domains, and report empirical results demonstrating improved learning speed over previous methods.

1. Introduction

Function approximation methods are commonly used in reinforcement learning (RL) for representing the value function in a compact form. Linear methods are particularly attractive due to their simplicity, but can be computationally inefficient when a sparse high-dimensional feature representation is used. *Piecewise linear function approximation* (PLFA), on the other hand, offers a potential advantage over traditional linear function approximation in terms of scalability. In the piecewise linear case, the value function is partitioned into a number of components, and updates need only consider a single component at any given time. The advantage of PLFA is that the computation time to do an update scales with the size of the feature vector in a partition, and not with the size of the feature vector over the entire space (as it does commonly in linear function approximation methods). This time complexity advantage should help PLFA apply to large problems.

However, one well-known issue with PLFA and function approximation methods in general is convergence. Algorithms that minimize the mean-square Bellman error, such as the residual gradient method (Baird, 1995), are guaranteed to converge but can suffer from poor performance. Other work has addressed this issue by employing methods, such as least squares or instrumental variables (Soderstrom and Stoica, 2002), that use second-order statistics to improve convergence time (Bradtke and Barto, 1996; Lagoudakis and Parr, 2003; Boyan, 2002; Xu, He, and Hu, 2002). Since these methods make more efficient use of samples, they promise to be effective. Unfortunately, they also rely on the assumption that the underlying process is stationary, and this assumption is easily violated in practice in at least three common cases: bootstrapping, change in the Markov Decision Process (MDP), and change in policy (Sutton and Barto, 1998). The result is that convergence again becomes slow. This slowdown is further compounded in PLFA because a shift in the value of one partition might result in a shift in the target value of a second partition.

We are interested in developing a fast algorithmic approach for RL with PLFA that will converge in all cases. We propose to handle the problem of non-stationarity by tracking the dynamics of the value function with techniques from linear system tracking. Modeling the nonstationarity as a random-walk, we show that the Kalman filter (Kalman, 1960) can be used to incrementally track changes in the value function. Since we assume a piecewise linear form, we can apply a separate Kalman filter to each component of the partitioned function. Although the Kalman filter is used for our examples, other methods from the linear system tracking literature could be used. While one of our assumptions is that the partitioning is supplied, we could also use variable resolution methods (Munos

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

and Moore, 2002; Potts and Sammut, 2005) to build a partitioning on the fly.

We cast our proposed technique as a general framework for RL methods that learn value functions, and show how popular algorithms such as TD, Q-Learning, MAXQ (Dietterich, 2000), and Sarsa fit in this framework. Although we do not provide a formal convergence proof here, our technique is a form of the residual gradient method and therefore should have similar convergence guarantees. We implemented the TD, Q-Learning and MAXQ algorithms in different domains and compared the performance of different linear function approximation methods. Our results suggest that tracking the dynamics of the value function can vastly improve convergence speed.

The paper is organized as follows. In Section 2, we review background material on linear function approximation and linear system tracking. We discuss related work in PLFA and present our general framework for RL with PLFA in Section 3. In Section 4, we define our dynamical model for RL with PLFA. We describe our experiments and report results in Section 5, and discuss conclusion and future work in Section 6.

2. Background

The problem of linear function approximation is a special case of linear system tracking. In linear system tracking, the goal is to generate an estimate $\hat{\mathbf{w}}(t) \in \Re^n$ of an unknown vector $\mathbf{w}(t) \in \Re^n$ through observations $\{y(1), \ldots, y(t)\}$. The dynamics of the system are modeled by the following equations:

$$\begin{aligned} \mathbf{w}(t) &= F(t-1)\mathbf{w}(t-1) + \boldsymbol{\mu}(t-1) \\ y(t) &= \mathbf{x}^T(t)\mathbf{w}(t) + \nu(t) \end{aligned}$$

where $F(t) \in \Re^{n \times n}$ is the transition matrix, $\mathbf{x}(t) \in \Re^n$ is the input vector, $\boldsymbol{\mu}(t) \in \Re^n$ is the process noise, and $\nu(t) \in \Re$ is the observation noise. It is commonly assumed that the noise processes $\{\boldsymbol{\mu}(t)\}$ and $\{\nu(t)\}$ are uncorrelated, white, and zero mean with covariances $\Sigma(t) \in \Re^{n \times n}$ and $\sigma^2(t) \in \Re$ respectively.

2.1. Linear Function Approximation

In linear function approximation, $\mathbf{w}$ is assumed to be stationary, that is $\forall t : F(t) = I$ and $\Sigma(t) = \mathbf{0}$. We begin with the least-mean-square (LMS) algorithm (Widrow and Hoff, 1960), which generates $\hat{\mathbf{w}}$ by minimizing

$$J(t) = \mathbb{E}\left[\left\|y(t) - \mathbf{x}^T(t)\hat{\mathbf{w}}(t)\right\|^2\right]$$

with respect to $\hat{\mathbf{w}}$ using gradient descent. Assuming $\frac{\partial y(t)}{\partial \hat{\mathbf{w}}} = 0$, the instantaneous error gradient is

$$\begin{aligned} \frac{\partial J(t)}{\partial \hat{\mathbf{w}}} &= 2\left(\frac{\partial y(t)}{\partial \hat{\mathbf{w}}} - \frac{\partial(\mathbf{x}^T(t)\hat{\mathbf{w}}(t))}{\partial \hat{\mathbf{w}}}\right)(y(t) - \mathbf{x}^T(t)\hat{\mathbf{w}}(t)) \\ &= -2\mathbf{x}(t)(y(t) - \mathbf{x}^T(t)\hat{\mathbf{w}}(t)). \end{aligned} \quad (1)$$

This gives the LMS update equation

$$\hat{\mathbf{w}}(t+1) = \hat{\mathbf{w}}(t) + \alpha\mathbf{x}(t)(y(t) - \mathbf{x}^T(t)\hat{\mathbf{w}}(t)) \quad (2)$$

where $\alpha \in (0, 1]$ is the learning rate. While the LMS algorithm is robust to uncertainties in the model, that is $\exists t : F(t) \neq I$ and $\Sigma(t) \neq \mathbf{0}$, it is often slow to converge (Haykin, 2001).

The least squares (LS) method generates $\hat{\mathbf{w}}$ by solving the equation $\hat{\mathbf{w}}(t) = P(t)\left[\frac{1}{t}\sum_{s=1}^{t}\frac{\mathbf{x}(s)y(s)}{\sigma^2(s)}\right]$ where $P(t) = \left[\frac{1}{t}\sum_{s=1}^{t}\frac{\mathbf{x}(s)\mathbf{x}^T(s)}{\sigma^2(s)}\right]^{-1}$. A forgetting factor $\lambda_{LS} \in (0, 1]$ can be introduced to decay the importance of older samples if there are uncertainties in the model. The recursive LS (RLS) algorithm computes $\hat{\mathbf{w}}(t)$ and $P(t)$ recursively using the following equations:

$$\begin{aligned} \hat{\mathbf{w}}^-(t) &= F(t-1)\hat{\mathbf{w}}^+(t-1) && (3) \\ P^-(t) &= F(t-1)P^+(t-1)F^T(t-1) + \Sigma(t-1) && (4) \\ \mathbf{k}(t) &= P^-(t)\mathbf{x}(t)(\mathbf{x}^T(t)P^-(t)\mathbf{x}(t) + \sigma^2(t))^{-1} && (5) \\ \hat{\mathbf{w}}^+(t) &= \hat{\mathbf{w}}^-(t) + \mathbf{k}(t)(y(t) - \mathbf{x}^T(t)\hat{\mathbf{w}}^-(t)) && (6) \\ P^+(t) &= (I - \mathbf{k}(t)\mathbf{x}^T(t))P^-(t) && (7) \end{aligned}$$

where $b^-(t)$ and $b^+(t)$ are the *a priori* and *a posteriori* estimate of the variable $b(t)$ respectively, $\mathbf{k}(t)$ is the gain vector, and $\forall t : F(t) = I$ and $\Sigma(t) = \mathbf{0}$. Although the LS method converges faster than the LMS algorithm, it is less robust to uncertainties in the model (Haykin, 2001).

If $\mathbf{x}(t)$ and $\nu(t)$ are correlated, the estimate $\hat{\mathbf{w}}(t)$ generated using the LS method will be biased. The instrumental variables (IV) method (Soderstrom and Stoica, 2002) is a generalization of the LS method that overcomes this by using an (user-defined) IV $\mathbf{z}(t) \in \Re^n$ that is correlated with $\mathbf{x}(t)$ but uncorrelated with $\nu(t)$. The IV method generates $\hat{\mathbf{w}}(t)$ by solving the equation $\hat{\mathbf{w}}(t) = P(t)\left[\frac{1}{t}\sum_{s=1}^{t}\frac{\mathbf{z}(s)y(s)}{\sigma^2(s)}\right]$ where $P(t) = \left[\frac{1}{t}\sum_{s=1}^{t}\frac{\mathbf{z}(s)\mathbf{x}^T(s)}{\sigma^2(s)}\right]^{-1}$. As with the LS method, a forgetting factor $\lambda_{IV} \in (0, 1]$ can be introduced to decay the importance of older samples if there are uncertainties in the model. The update equations for recursive computation of $\hat{\mathbf{w}}(t)$ and $P(t)$ using the IV method are similar to that of RLS differing only in equation (5), which is replaced with the following equation:

$$\mathbf{k}(t) = P^-(t)\mathbf{z}(t)(\mathbf{x}^T(t)P^-(t)\mathbf{z}(t) + \sigma^2(t))^{-1}. \quad (8)$$

Due to its close relation with the RLS algorithm, we will refer to this as the RLS-IV algorithm.

2.2. Linear System Tracking

For the general case where $\mathbf{w}$ is nonstationary, that is $\exists t : F(t) \neq I$ or $\Sigma(t) \neq \mathbf{0}$, the Kalman filter is the optimal linear mean-square estimator (Simon, 2006). Its update equations are those of the RLS algorithm, equations (3) to (7), except that no conditions are placed on $F(t)$ or $\Sigma(t)$.

Other linear system tracking algorithms include the H_∞ filter and the particle filter. See Simon's (2006) book for an extended discussion of these algorithms.

3. Reinforcement Learning with Piecewise Linear Function Approximation

An RL problem is commonly formalized as an MDP. An MDP is a four-tuple $(S, A, T^{a_t}_{s_t,s_{t+1}}, R^{a_t}_{s_t,s_{t+1}})$ where S is the set of states, A is the set of actions, $T^{a_t}_{s_t,s_{t+1}}$ is the state transition probability of reaching $s_{t+1} \in S$ after executing $a_t \in A$ in $s_t \in S$, and $r_t = R^{a_t}_{s_t,s_{t+1}}$ is the reward received for reaching $s_{t+1} \in S$ after executing $a_t \in A$ in $s_t \in S$.

The goal of RL is to find some policy $\pi : S \to A$ that maximizes $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where $\gamma \in (0, 1]$ is the discount factor. This is commonly achieved indirectly by learning either the state value function $V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]$ or the state action value function $Q(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right]$. Defining $x \mapsto y$ as the updating of x towards y and using the substitutions in Table 1, value function learning algorithms such as TD, Sarsa, and Q-Learning can be generalized as

$$f(\omega_t) \mapsto r_t + \gamma f(\omega'_t) \tag{9}$$

where $f : \Omega \to \Re$ is the target value function, $\omega_t \in \Omega$ is the update point, and $\omega'_t \in \Omega$ is the target point. If the target value function is V then $\Omega = S$, and if the target function is Q then $\Omega = S \times A$.

Table 1. Relationships between value function learning algorithms and the generalized update equation.

Algorithm	f	ω_t	ω'_t
TD	V	s_t	s_{t+1}
Sarsa	Q	(s_t, a_t)	(s_{t+1}, a_{t+1})
Q-Learning	Q	(s_t, a_t)	$\left(s_{t+1}, \underset{a' \in A}{\operatorname{argmax}} Q(s_{t+1}, a')\right)$

With equation (9), our general framework for RL with PLFA is constructed as follows. We begin by partitioning Ω into m partitions $\Omega_1, \ldots \Omega_m$ such that $\Omega = \bigcup_{i=1}^{m} \Omega_i$ and $\forall i, j : i \neq j \to \Omega_i \cap \Omega_j = \emptyset$. Next we define a function $\psi : \Omega \to \{1, \ldots, m\}$ that maps $\omega \in \Omega$ to the index of the partition to which it belongs. We also define for each partition Ω_i a parameter vector $\hat{\mathbf{w}}_i \in \Re^n$, and a function $\phi_i : \Omega_i \to \Re^n$ that maps $\omega \in \Omega_i$ to a feature vector. Then we approximate f as a linear function of the feature vector and the parameter vector, that is $f(\omega) \approx \mathbf{x}_\omega^T \hat{\mathbf{w}}_\omega$ where we write $\hat{\mathbf{w}}_\omega = \hat{\mathbf{w}}_{\psi(\omega)}$ and $\mathbf{x}_\omega = \phi_{\psi(\omega)}(\omega)$ for brevity. Hence in this framework, the state value function $V(s) \approx \mathbf{x}_s^T \hat{\mathbf{w}}_s$, and the state action value function $Q(s, a) \approx \mathbf{x}_{s,a}^T \hat{\mathbf{w}}_{s,a}$.

3.1. Previous Work in RL with PLFA

Using the framework for RL with PLFA, equation (9) can now be written as

$$\mathbf{x}_{\omega_t}^T \hat{\mathbf{w}}_{\omega_t} \mapsto r_t + \gamma \mathbf{x}_{\omega'_t}^T \hat{\mathbf{w}}_{\omega'_t}.$$

This update rule can be derived by minimizing

$$J_d(t) = \mathbb{E}\left[\left\|(r_t + \gamma \mathbf{x}_{\omega'_t}^T \hat{\mathbf{w}}_{\omega'_t}(t)) - \mathbf{x}_{\omega_t}^T \hat{\mathbf{w}}_{\omega_t}(t)\right\|^2\right] \tag{10}$$

using the LMS algorithm (Section 2.1). Substituting $\hat{\mathbf{w}}(t) = \hat{\mathbf{w}}_{\omega_t}(t)$, $\mathbf{x}(t) = \mathbf{x}_{\omega_t}$, and $y(t) = r_t + \gamma \mathbf{x}_{\omega'_t}^T \hat{\mathbf{w}}_{\omega'_t}(t)$ into equation (2), we get the standard update equation (Perkins and Precup, 2002; Sutton and Barto, 1998; Tsitsiklis and Van Roy, 1997) for RL with PLFA,

$$\hat{\mathbf{w}}_{\omega_t}(t+1) = \hat{\mathbf{w}}_{\omega_t}(t) + \alpha \mathbf{x}_{\omega_t}(r_t + \gamma \mathbf{x}_{\omega'_t}^T \hat{\mathbf{w}}_{\omega'_t}(t) - \mathbf{x}_{\omega_t}^T \hat{\mathbf{w}}_{\omega_t}(t)). \tag{11}$$

Note that if $m = |\Omega|$ and $\forall i : \phi_i(\omega) = 1$, we get the standard update equation for tabular form RL (Sutton and Barto, 1998). As with LMS, equation (11) is often slow to converge; this can be improved by using the same set of substitutions in the LS method (Choi and Van Roy, 2006).

While the convergence of equation (11) in the tabular case, that is $m = |\Omega|$, is well known, convergence in the general case, that is $m \neq |\Omega|$, has been in doubt since Baird's (1995) work. The reason is the assumption $\frac{\partial y(t)}{\partial \hat{\mathbf{w}}} = 0$ made in the derivation of LMS (equation (1)) is no longer true if $\psi(\omega_t) = \psi(\omega'_t) = i$ since

$$\frac{\partial y(t)}{\partial \hat{\mathbf{w}}} = \frac{\partial (r_t + \gamma \mathbf{x}_{\omega'_t}^T \hat{\mathbf{w}}_{\omega'_t}(t))}{\partial \hat{\mathbf{w}}_{\omega_t}} = \gamma \mathbf{x}_{\omega'_t}$$

Hence if $\psi(\omega_t) = \psi(\omega'_t) = i$, the correct update equation should be:

$$\hat{\mathbf{w}}_i(t+1) = \hat{\mathbf{w}}_i(t) + \alpha(\mathbf{x}_{\omega_t} - \gamma \mathbf{x}_{\omega'_t})(r_t + \gamma \mathbf{x}_{\omega'_t}^T \hat{\mathbf{w}}_i(t) - \mathbf{x}_{\omega_t}^T \hat{\mathbf{w}}_i(t)). \tag{12}$$

As noted by Baird (1995), equation (12) converges slower than equation (11) in some circumstances. To overcome this, he proposed a method of combining the two update equations parametrically.

We can also arrive at equation (12) by minimizing the equivalent of equation (10),

$$J_{rg}(t) = \mathbb{E}\left[\left\|r_t - (\mathbf{x}_{\omega_t} - \gamma\mathbf{x}_{\omega'_t})^T\hat{\mathbf{w}}_i(t)\right\|^2\right]$$

using LMS with substitutions $\hat{\mathbf{w}}(t) = \hat{\mathbf{w}}_i(t)$, $\mathbf{x}(t) = (\mathbf{x}_{\omega_t} - \gamma\mathbf{x}_{\omega'_t})$, and $y(t) = r_t$. Using these substitutions, Bradtke and Barto (1996) and Lagoudakis and Parr (2003) used the IV method, with $\mathbf{x}_{\omega_t}$ as the IV, to improve the speed of convergence for the case where $m = 1$. Boyan (2002) and Xu, He, and Hu (2002) extended their work by using $\sum_{i=0}^{t}\lambda_{RL}^{t-i}\mathbf{x}_{\omega_i}$ as the IV where $\lambda_{RL} \in [0,1]$ is the eligibility factor. However, these algorithms can be computationally inefficient if for any $\omega \in \Omega$, only a small number of entries in $\mathbf{x}_\omega$ are non-zero. Geramifard, Bowling, and Sutton (2006) proposed an approximation algorithm that improves the computational efficiency in this case.

4. Dynamical Model for RL with PLFA

From the linear system tracking perspective (Section 2), the goal of RL with PLFA is to compute for each partition i, $i = 1, \ldots, m$, an estimate $\hat{\mathbf{w}}_i \in \Re^n$ of the unknown vector $\mathbf{w}_i \in \Re^n$ through observations $\{y_i(1) \ldots y_i(t)\}$. The dynamics of the systems are modeled by the following equations:

$$\mathbf{w}_i(t) = F_i(t-1)\mathbf{w}_i(t-1) + \boldsymbol{\mu}_i(t-1) \quad (13)$$
$$y_i(t) = \mathbf{x}_i^T(t)\mathbf{w}_i(t) + \nu_i(t) \quad (14)$$

where $F_i(t) \in \Re^{n\times n}$ is the transition matrix, $\mathbf{x}_i(t) \in \Re^n$ is the input vector, $\boldsymbol{\mu}_i(t) \in \Re^n$ is the process noise, and $\nu_i(t) \in \Re$ is the observation noise. It will be assumed that the noise processes $\{\boldsymbol{\mu}_i(t)\}$ and $\{\nu_i(t)\}$ are uncorrelated, white, and zero mean with covariances $\Sigma_i(t) \in \Re^{n\times n}$ and $\sigma_i^2(t) \in \Re$ respectively.

In RL with PLFA, $\nu_i(t)$ models (i) the approximation error, that is the difference between the function and the linear model used, and (ii) the uncertainty in the value as a result of the state transition probability function $T^{a_t}_{s_t,s_{t+1}}$. Because of these $\nu_i(t)$ might be correlated with $x_i(t)$.

When the update point ω_t is not in partition i, that is $\psi(\omega_t) \neq i$, there are no inputs to and observations from system i. This is modeled by setting $\mathbf{x}_i(t)$ and $y_i(t)$ to zero, that is $\mathbf{x}_i(t) = \mathbf{0}$ and $y_i(t) = 0$, whenever $\psi(\omega_t) \neq i$. This will also force $\nu_i(t)$ to be 0.

An interesting issue arises when $\psi(\omega_t) = \psi(\omega'_t) = i$, $\mathbf{x}_{\omega_t} - \gamma\mathbf{x}_{\omega'_t} = 0$, and $r_t \neq 0$. From equation (14) and the substitutions used in the previous section where $\psi(\omega_t) = \psi(\omega'_t) = i$, that is $\mathbf{x}_i(t) = \mathbf{0}$ and $y_i(t) = r_t$, we get $r_t = \nu_i(t)$. But this situation is also possible in cases where there are no approximation errors and the transition is deterministic, that is $\nu_i(t) = 0$. This is inconsistent as $r_t \neq 0$. An example of this situation occurring is a self-transition as a result of a collision with an obstacle in a navigation problem when $\gamma = 1$. To avoid this, we will learn the value of ω_t instead of the reward of the transition from ω_t to ω'_t, that is we will revert to the substitutions used when $\psi(\omega_t) \neq \psi(\omega'_t)$ if $\psi(\omega_t) = \psi(\omega'_t) = i$, $\mathbf{x}_{\omega_t} - \gamma\mathbf{x}_{\omega'_t} = 0$, and $r_t \neq 0$. With these and the substitutions used in the preceding section, $\mathbf{x}_i(t)$, $y_i(t)$, and $\nu_i(t)$ are defined by equations (15) to (17) respectively.

Recall that for the LMS, LS and IV algorithms, $\mathbf{w}_i$ is assumed to be stationary, that is $\forall t : F_i(t) = I$ and $\Sigma_i(t) = \mathbf{0}$. However it is known (Sutton and Barto, 1998) that this assumption can be violated by either (i) the use of bootstrapping methods, (ii) a change in policy, or (iii) a change in the MDP. It is trivial to construct an example with deterministic transition and no approximation error, that is $\nu_i(t) = 0$, for each case where $\exists j, k : \omega_j = \omega_k$, $i = \psi(\omega_j)$, and $y_i(j) \neq y_i(k)$. We can conclude in such an example that $\mathbf{w}_i(j) \neq \mathbf{w}_i(k)$, hence $\exists l$ between j and k such that $F_i(l) \neq I$ or $\Sigma_i(l) \neq \mathbf{0}$.

While LMS is robust to uncertainties in $F_i(t)$ and $\Sigma_i(t)$, it is slow to converge. The LS and IV methods on the other hand converge faster than LMS but are less robust in tracking the changes in $\mathbf{w}_i$ due to a mismatch in the model. We will attempt to reduce this mismatch through the use of the Kalman filter.

Our first task in applying the Kalman filter is to find values for $F_i(t)$ and $\Sigma_i(t)$. It is difficult or impossible to determine the true values of $F_i(t)$ and $\Sigma_i(t)$, since, among other things, knowledge of (i) the form of the approximated value function, (ii) the trajectory of ω_t, (iii) the policy used, and (iv) the underlying MDP will be required. As there is no reason to believe that $\mathbf{w}_i$ will travel in any particular direction, we will model the transition of $\mathbf{w}_i$ as a random walk, that is $\forall t : F_i(t) = I$, where the value of $\Sigma_i(t)$ is domain-dependent and nonstationary in general.

Finally, we note that from equation (13), the value of $\mathbf{w}_i(t)$ can change even though the update point ω_t is not in partition i, as it is dependent only on the values of $F_i(t)$ and $\boldsymbol{\mu}_i(t)$. To reduce the computation effort required, we let $\boldsymbol{\mu}_i(t)$ be defined by equation (18) where $\Sigma_i(t)$ is now the covariance of the transition

Table 2. Dynamical Model for RL with PLFA.

$$\mathbf{w}_i(t) = \mathbf{w}_i(t-1) + \boldsymbol{\mu}_i(t-1)$$

$$\mathbf{x}_i(t) = \begin{cases} \mathbf{0} & \text{IF } i \neq \psi(\omega_t) \\ \mathbf{x}_{\omega_t} - \gamma \mathbf{x}_{\omega'_t} & \text{IF } i = \psi(\omega_t) = \psi(\omega'_t) \text{ AND} \\ & (\mathbf{x}_{\omega_t} - \gamma \mathbf{x}_{\omega'_t} \neq 0 \text{ OR } r_t = 0) \\ \mathbf{x}_{\omega_t} & \text{OTHERWISE} \end{cases} \tag{15}$$

$$y_i(t) = \mathbf{x}_i^T(t)\mathbf{w}_i(t) + \nu_i(t)$$

$$= \begin{cases} 0 & \text{IF } i \neq \psi(\omega_t) \\ r_t & \text{IF } i = \psi(\omega_t) = \psi(\omega'_t) \text{ AND} \\ & (\mathbf{x}_{\omega_t} - \gamma \mathbf{x}_{\omega'_t} \neq 0 \text{ OR } r_t = 0) \\ r_t + \gamma \mathbf{x}_{\omega'_t}^T \hat{\mathbf{w}}_{\omega'_t}(t) & \text{OTHERWISE} \end{cases} \tag{16}$$

$$\nu_i(t) = \begin{cases} 0 & \text{IF } i \neq \psi(\omega_t) \\ X \sim N(0, \sigma_i^2(t)) & \text{OTHERWISE} \end{cases} \tag{17}$$

$$\boldsymbol{\mu}_i(t) = \begin{cases} 0 & \text{IF } i \neq \psi(\omega_t) \\ X \sim N(0, \Sigma_i(t)) & \text{OTHERWISE} \end{cases} \tag{18}$$

from $\mathbf{w}_i(t)$ to $\mathbf{w}_i(t')$, and t' is the next time of visit to partition i, that is $t' = \text{argmin}_{j>i}(i = \psi(\omega_j))$. With this, the Kalman filter (equations (3) to (7)) can now be used to incrementally track the m systems modeled by the equations in Table 2 and update $\hat{\mathbf{w}}_i(t)$ for partition i only when $i = \psi(\omega_t)$.

5. Experimental Results

In this section, we present experimental results in three domains: the *hop world* domain, the *mountain car* domain and the *continuous grid world* domain. Using the model described in the previous section, we compare and analyze the performance of four linear approximation algorithms in these domains. Because we are interested specifically in problems that would require a prohibitively large feature vector, we chose to consider only piecewise linear value function representations as opposed to the original linear representations used in the competing algorithms.

The first linear approximation algorithm we consider is the LMS algorithm. To avoid further slowdown in the learning speed, we ignore the case where $\psi(\omega_t) = \psi(\omega'_t)$ and only use equation (11). As with Baird (1995), we will refer to this as the direct algorithm. The learning rate α for this algorithm will be 0.1. This value and the values for other parameters in the experiments are commonly used values for their respective algorithms.

The second is the RLS-IV algorithm. As with Bradtke and Barto (1996), we will use $\mathbf{x}_{\omega_t}$ as the IV, and refer to this as the LSTD algorithm. We will also use $\lambda_{IV} = 0.995$ as the forgetting factor.

The third is the RLS algorithm. A forgetting factor $\lambda_{LS} = 0.995$ will also be used for this algorithm.

The last is the Kalman filter. Although the noise covariances $\sigma_i^2(t) = 1$ and $\Sigma_i(t) = \beta I$ are domain-dependent and nonstationary in general, we will make a simplifying assumption in our experiments that they are constant, that is we will set $\forall t : \sigma_i^2(t) = 1$ and $\Sigma_i(t) = \beta I$. Unless otherwise stated, $\beta = 0.05$. As with Haykin (2001), we will refer to this as the extended RLS algorithm (ERLS).

In the experiments, the discount factor γ is set to 1. The ε-greedy action selection strategy, where $\varepsilon = 0.1$, is used when learning, and the greedy action selection policy will be used otherwise.

All results presented are the average of 100 trials. The results from the first experiment were collected at every episode while learning. The results from the other two experiments were collected at every sixth episode with learning disabled.

5.1. Boyan's Hop World

The first experiment is Boyan's (2002) hop world domain. This is a 13-state Markov chain where state s^0 is an absorbing state, state s^1 transits to state s^0 with probability 1 and a reward of -2, and state s^i, $2 \leq i \leq 12$, transits to either s^{i-1} or s^{i-2}, each with a probability of 0.5 and a reward of -3.

In this experiment, the TD algorithm will be used to learn the state value function V. With $\Omega = S$ and $m = 1$, the feature vectors for states s^{12}, s^8, s^4, s^0 are $[1,0,0,0]$, $[0,1,0,0]$, $[0,0,1,0]$, and $[0,0,0,1]$ respectively. The feature vectors for other states are obtained by linearly interpolating between these.

To simulate a change in the MDP, and hence in $\mathbf{w}$, the sign of the rewards will be inverted from the 150^{th} episode onwards. Hence the true value of state s^i is $-2i$ before the inversion and $2i$ after that.

Figure 1 shows the root-mean-square error (RMSE) between the learned values of the states and their true values for all four algorithms. As seen in the figure, the direct and LSTD algorithms converge to values close to the true values of the states, while the RLS and ERLS algorithms converge to a biased result. This is to be expected since $\nu_i(t)$ is correlated with $x_i(t)$ due to the uncertainty in the values as a result of the state transition probability function $T^{a_t}_{s_t, s_{t+1}}$.

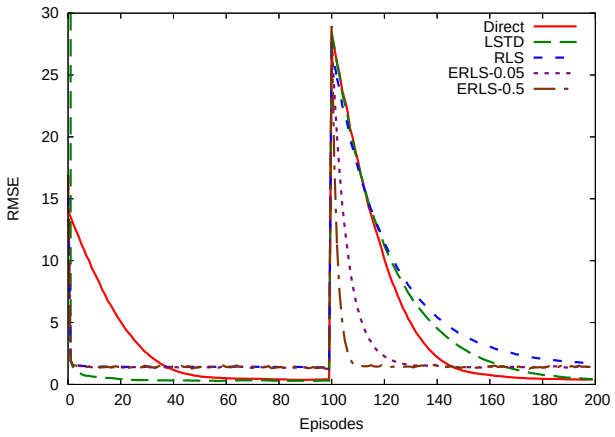

Figure 1. Root-mean-square error (RMSE) between the learned values of the states and their true values in Boyan's Hop World domain.

Also seen in Figure 1, the LSTD algorithm converges much faster than the direct algorithm initially, but after the switch in the sign of the reward in the 150^{th} episode, the LSTD algorithm converges slower than the direct algorithm. Its performance after the inversion is comparable to that of the RLS algorithm. On the other hand, both instances of the ERLS algorithm, where $\beta = 0.05$ for one and $\beta = 0.5$ for the other, converges much faster than the direct algorithm before and after the inversion. This is due to the incorrect assumption that the unknown vector $\mathbf{w}$ is stationary, made by both the IV and LS methods.

As a larger β value results in faster convergence speed, β can be viewed as the equivalent of the learning rate α in LMS algorithms.

5.2. Mountain Car

The second experiment is Sutton and Barto's (1998) mountain car domain. In this domain, the state space $S = \{(u, \dot{u}) \mid u \in (-1.2, 0.5), \dot{u} \in (-0.7, 0.7)\}$, the action space $A = \{-1, 0, 1\}$, and $\forall t$ the reward $r_t = -1$. The next state $(u_{t+1}, \dot{u}_{t+1}) \in S$ after executing $a_t \in A$ in $(u_t, \dot{u}_t) \in S$ is determined by the following equations:

$$u_{t+1} = \text{bound}[u_t + u_{t+1}]$$
$$\dot{u}_{t+1} = \text{bound}[\dot{u}_t + 0.001a_t - 0.0025\cos(3u_t)]$$

where the bound operation enforces the constraints on S. If the left bound of u is reached, $\dot{u}$ will be set to zero. Each episode begins in $(-\frac{\pi}{6}, 0) \in S$ and terminates after 2500 steps. The goal in this domain is to reach the right bound of u, at which time the episode will also be terminated.

In this experiment, the Q-Learning algorithm is used to learn the state action value function, hence $\Omega = S \times A$. To represent $(s, a) \in S \times A$ as a feature vector, S is first divided up by a 16×16 grid. Then for each block in the grid, a partition is created for each action, thus giving $m = 16 \times 16 \times 3$ partitions. Letting b^L and b^S be, respectively, the left bound and the size of the dimension for variable b in the block $s = (u, \dot{u}) \in S$ is in, the feature vector $\mathbf{x}_{s,a} = (1, \frac{u - u^L}{u^S}, \frac{\dot{u} - \dot{u}^L}{\dot{u}^S})$.

Figure 2 shows the number of steps taken to reach the goal for all four algorithms. As expected, the ERLS algorithm converges faster than the direct algorithm, which in turn converges faster than the RLS and LSTD algorithms. Curiously though, the RLS algorithm performs significantly better than the LSTD algorithm. One possible reason for this could be that the IV being used is inappropriate.

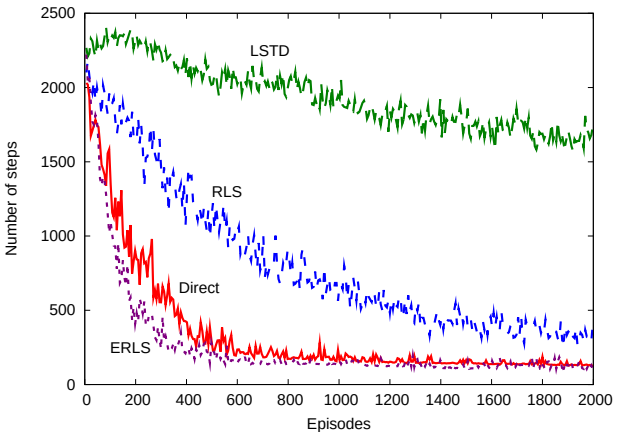

Figure 2. Number of steps taken to reach the goal in the mountain car domain.

As with the previous experiment, the poor performance by the RLS and LSTD algorithms is due to the stationary assumption made by both the IV and LS methods. Unlike the previous experiment, the resulting slowdown in learning speed is further compounded by the fact that a shift in $\mathbf{w}_i$ of partition i might result in a shift in $\mathbf{w}_j$ of any partition j where $\exists t : \omega_t \in \Omega_j$ and $\omega'_t \in \Omega_i$. While it is possible that the RLS and LSTD algorithms could have a better performance if a similar single partition representation is used, that is $n = 16 \times 16 \times 3 \times 3$ and $m = 1$, the computational cost of such representation would be prohibitive.

From Figure 2, one can also see that results for the ERLS algorithm have a noisy tail. This issue could be resolved with an appropriate annealing schedule for β.

5.3. Continuous Grid World

The last experiment is a continuous version of the grid world domain. In this domain, the state space $S = \{(u, v, \theta) \mid u, v \in (0, 2), \theta \in [0°, 360°)\}$, the action space $A = \{(-1, 0), (1, 0), (0, -1), (0, 1)\}$, and $\forall t$ the reward $r_t = -1$. The next state $(u_{t+1}, v_{t+1}, \theta_{t+1}) \in S$ after executing $(b_t, c_t) \in A$ in $(u_t, v_t, \theta_t) \in S$ is determined by the following equations:

$$\begin{aligned} u_{t+1} &= \text{bound}[u_t + 0.05 b_t \cos(\theta_t)] \\ v_{t+1} &= \text{bound}[v_t + 0.05 b_t \sin(\theta_t)] \\ \theta_{t+1} &= (\theta_t + 5 c_t) \bmod 360. \end{aligned}$$

Each episode begins in $(0, 0, 0) \in S$ and terminates either after 5000 steps or when the goal is reached. The goal in this domain is to reach a region bounded by $1.6 < u < 1.7$ and $1.4 < v < 1.5$.

In this experiment, we use Dietterich's (2000) MAXQ algorithm to learn the value function. The relationship of MAXQ's update equation to the generalized update equation (equation (9)) is similar to that of the other value function learning algorithms. A two-level task hierarchy (Figure 3) with four navigation subtasks and a root subtask is used. A navigation subtask nav_i, $i = 0, \ldots, 3$, has $S_{\text{nav}_i} = \{\theta \mid \theta \in [0°, 360°)\}$ and A as its state space and action space respectively, and terminates if $(0, 1) \in A$ is executed in $\theta \in S_{\text{nav}_i}$ and $i \times 90° - 2.5 < \theta < i \times 90° + 2.5$. The root subtask has $S_{\text{root}} = \{(u, v) \mid u, v \in (0, 2)\}$ and $A_{\text{root}} = \{\text{nav}_i \mid i = 0, \ldots, 3\}$ as its state space and action space respectively, and terminates whenever the goal region of the domain is reached.

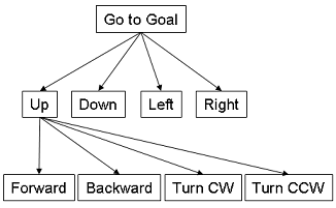

Figure 3. Task hierarchy for continuous grid world domain.

The feature vector used in this experiment is similar to that used in the previous one. A 16×16 grid is used to partition S_{root}, and each S_{nav_i} is divided into 16 equal-length nonintersecting segments. Then for each block in the grid of each subtask, a partition is created for each action in that subtask, thus giving $m = 16 \times 16 \times 4 + 16 \times 4 \times 4$ partitions. Using the definitions in the previous experiment, the feature vector $\mathbf{x}_{\text{root},s,a}$ for $(s, a) \in S_{\text{root}} \times A_{\text{root}}$ in the root subtask is $(1, \frac{u - u^L}{u^S}, \frac{v - v^L}{v^S})$, and the feature vector $\mathbf{x}_{\text{nav}_i,s,a}$ for $(s, a) \in S_{\text{nav}_i} \times A$ in the navigation subtask is $(1, \frac{\theta - \theta^L}{\theta^S})$.

Figure 4 shows the number of steps taken to reach the goal for the direct and ERLS algorithms. The results of the RLS and LSTD algorithms are omitted because both fail to reach the goal most of the time. As seen in the figure, the ERLS algorithm outperforms the direct algorithm once again.

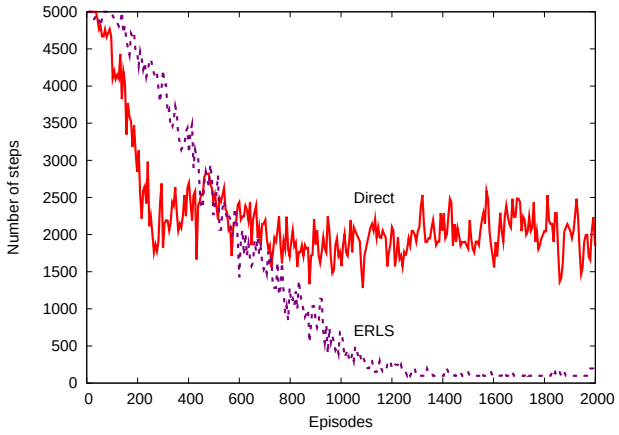

Figure 4. Number of steps taken to reach the goal in the continuous grid world domain.

6. Conclusion and Future Work

The advantages of piecewise linear over linear function approximation lie in problems where a sparse high-dimensional feature vector is used, since computational requirements are proportional to the number of features in a partition. In PLFA, as we have noted, a change in the value of an arbitrary partition induced by (i) the use of bootstrapping methods, (ii) a change in policy, or (iii) a change in the MDP, can result in a change in the value of a second arbitrary partition. Our empirical results support our claim that modeling the value function dynamics as a random-walk and then tracking the value function using the Kalman filter can improve convergence speed significantly. Speedup is more evident when more partitions are used, as seen in the continuous grid world experiment.

One drawback of our method reflected in the experimental results is that our approximation is biased. While the IV method can remove this bias, the use of an inappropriate IV, as seen in our results, can substantially impair its performance. The search for an appropriate IV is thus an issue for further research.

Another important idea to explore is variable resolution representation (Munos and Moore, 2002; Potts and Sammut, 2005). Given an initial partitioning, we can refine one partition without recomputing the value estimates of other partitions from scratch. We could then eliminate dependence on a good initial partitioning and increase the applicability of our method in complex domains.

With the link between RL with PLFA and linear system tracking established, the extensive linear system tracking literature can be further exploited. We expect that other ideas and algorithms from this body of work can be fruitfully adapted for use in RL with PLFA.

Acknowledgments

We would like to thank Alan Blair, Will Uther, Bernhard Hengst, other NICTA researchers, and the anonymous reviewers for their helpful comments. NICTA is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs.

References

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. *Twelfth International Conference on Machine Learning* (pp. 30–37). San Francisco: Morgan Kaufman Publishers.

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

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

Choi, D., & Roy, B. V. (2006). A generalized kalman filter for fixed point approximation and efficient temporal difference learning. *Discrete Event Dynamic Systems*, *16*, 207 – 239.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

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

Haykin, S. (2001). *Adaptive filter theory (4th edition)*. Prentice Hall.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, *82*, 35–45.

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *The Journal of Machine Learning Research*, *4*, 1107 – 1149.

Munos, R., & Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning*, *49*, 291–323.

Perkins, T. J., & Precup, D. (2002). A convergent form of approximate policy iteration. *Neural Information Processing Systems* (pp. 1595–1602). Vancouver, British Columbia, Canada: MIT Press.

Potts, D., & Sammut, C. (2005). Incremental learning of linear model trees. *Machine Learning*, *61*, 5–48.

Simon, D. (2006). *Optimal state estimation: Kalman, H-infinity, and nonlinear approaches*. Wiley-Interscience.

Soderstrom, T., & Stoica, P. (2002). Instrumental variable methods for system identification. *Circuits Systems Signal Processing*, *21*, 1–9.

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

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

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. *IRE WESCON Convention Record* (pp. 96–104). New York.

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