Predictive State Temporal Difference Learning

Byron Boots
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
beb@cs.cmu.edu

Geoffrey J. Gordon
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
ggordon@cs.cmu.edu

Abstract

We propose a new approach to value function approximation which combines linear temporal difference reinforcement learning with subspace identification. In practical applications, reinforcement learning (RL) is complicated by the fact that state is either high-dimensional or partially observable. Therefore, RL methods are designed to work with *features* of state rather than state itself, and the success or failure of learning is often determined by the suitability of the selected features. By comparison, subspace identification (SSID) methods are designed to *select* a feature set which preserves as much information as possible about state. In this paper we connect the two approaches, looking at the problem of reinforcement learning with a large set of features, each of which may only be marginally useful for value function approximation. We introduce a new algorithm for this situation, called Predictive State Temporal Difference (PSTD) learning. As in SSID for predictive state representations, PSTD finds a linear *compression operator* that projects a large set of features down to a small set that preserves the maximum amount of predictive information. As in RL, PSTD then uses a Bellman recursion to estimate a value function. We discuss the connection between PSTD and prior approaches in RL and SSID. We prove that PSTD is statistically consistent, perform several experiments that illustrate its properties, and demonstrate its potential on a difficult optimal stopping problem.

1 Introduction

We wish to estimate the value function of a policy in an unknown decision process in a high dimensional and partially-observable environment. We represent the value function in a *linear architecture*, as a linear combination of *features* of (sequences of) observations. A popular family of learning algorithms called temporal difference (TD) methods [1] are designed for this situation. In particular, least-squares TD (LSTD) algorithms [2, 3, 4] exploit the linearity of the value function to estimate its parameters from sampled trajectories, i.e., from sequences of feature vectors of visited states, by solving a set of linear equations.

Recently, Parr et al. looked at the problem of value function estimation from the perspective of both model-free and model-based reinforcement learning [5]. The model-free approach (which includes TD methods) estimates a value function directly from sample trajectories. The model-based approach, by contrast, first learns a model of the process and then computes the value function from the learned model. Parr et al. demonstrated that these two approaches compute exactly the same value function [5]. In the current paper, we build on this insight, while simultaneously finding a compact set of features using powerful methods from system identification.

First, we look at the problem of improving LSTD from a model-free predictive-bottleneck perspective: given a large set of features of history, we devise a new TD method called Predictive State Temporal Difference (PSTD) learning. PSTD estimates the value function through a bottleneck that preserves only *predictive* information (Section 3). Second, we look at the problem of value function estimation from a model-based perspective (Section 4). Instead of learning a linear transition model in feature space, as in [5], we use subspace identification [6, 7] to learn a PSR from our samples. Since PSRs are at least as compact as POMDPs, our representation can naturally be viewed as a value-directed compression of a much larger POMDP. Finally, we show that our two improved methods are equivalent. This result yields some appealing theoretical benefits: for example, PSTD features can be explicitly interpreted as a statistically consistent estimate of the true underlying system state. And, the feasibility of finding the true value function can be shown to depend on the *linear dimension* of the dynamical system, or equivalently, the dimensionality of the predictive state representation—*not* on the cardinality of the POMDP state space. Therefore our representation is naturally "compressed" in the sense of [8], speeding up convergence.

We demonstrate the practical benefits of our method with several experiments: we compare PSTD to competing algorithms on a synthetic example and a difficult optimal stopping problem. In the latter problem, a significant amount of prior work has gone into hand-tuning features. We show that, if we add a large number of weakly relevant features to these hand-tuned features, PSTD can find a predictive subspace which performs much better than competing approaches, improving on the best previously reported result for this problem by a substantial margin. The theoretical and empirical results reported here suggest that, for many applications where LSTD is used to compute a value function, PSTD can be simply substituted to produce better results.

2 Value Function Approximation

We start from a discrete time dynamical system with a set of states $\mathcal{S}$, a set of actions A, a distribution over initial states π_0, a transition function T, a reward function $\mathcal{R}$, and a discount factor $\gamma \in [0, 1]$. We seek a policy π, a mapping from states to actions. For a given policy π, the value of state s is defined as the expected discounted sum of rewards when starting in state s and following policy π, $J^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t) \mid s_0 = s, \pi\right]$. The value function obeys the Bellman equation

$$J^\pi(s) = \mathcal{R}(s) + \gamma \textstyle\sum_{s'} J^\pi(s') \Pr[s' \mid s, \pi(s)] \tag{1}$$

If we know the transition function T, and if the set of states $\mathcal{S}$ is sufficiently small, we can find an optimal policy with *policy iteration*: pick an initial policy π, use (1) to solve for the value function J^π, compute the *greedy policy* for J^π (setting the action at each state to maximize the right-hand side of (1)), and repeat. However, we consider instead the harder problem of estimating the value function when s is a *partially observable* latent variable, and when the transition function T is unknown. In this situation, we receive information about s through observations from a finite set O. We can no longer make decisions or predict reward based on $\mathcal{S}$, but instead must use a *history* (an ordered sequence of action-observation pairs $h = a_1^h o_1^h \dots a_t^h o_t^h$ that have been executed and observed prior to time t): $\mathcal{R}(h)$, $J(h)$, and $\pi(h)$ instead of $\mathcal{R}(s)$, $J^\pi(s)$, and $\pi(s)$. Let $\mathcal{H}$ be the set of all possible histories. $\mathcal{H}$ is often very large or infinite, so instead of finding a value separately for each history, we focus on value functions that are linear in *features* of histories

$$J^\pi(h) = w^\mathsf{T} \phi^\mathcal{H}(h) \tag{2}$$

Here $w \in \mathbb{R}^j$ is a parameter vector and $\phi^\mathcal{H}(h) \in \mathbb{R}^j$ is a feature vector for a history h. So, we can rewrite the Bellman equation as

$$w^\mathsf{T} \phi^\mathcal{H}(h) = \mathcal{R}(h) + \gamma \textstyle\sum_{o \in O} w^\mathsf{T} \phi^\mathcal{H}(h\pi o) \Pr[h\pi o \mid h\pi] \tag{3}$$

where $h\pi o$ is history h extended by taking action $\pi(h)$ and observing o.

2.1 Least Squares Temporal Difference Learning

In general we don't know the transition probabilities $\Pr[h\pi o \mid h]$, but we do have samples of state features $\phi_t^\mathcal{H} = \phi^\mathcal{H}(h_t)$, next-state features $\phi_{t+1}^\mathcal{H} = \phi^\mathcal{H}(h_{t+1})$, and immediate rewards $\mathcal{R}_t = \mathcal{R}(h_t)$. We can thus *estimate* the Bellman equation

$$w^\mathsf{T} \phi_{1:k}^\mathcal{H} \approx \mathcal{R}_{1:k} + \gamma w^\mathsf{T} \phi_{2:k+1}^\mathcal{H} \tag{4}$$

(Here we have used $\phi_{1:k}^\mathcal{H}$ to mean the matrix whose columns are $\phi_t^\mathcal{H}$ for $t = 1 \dots k$.) We can can immediately attempt to estimate the parameter w by solving this linear system in the least squares sense: $\hat{w}^{\mathsf{T}} = \mathcal{R}_{1:k}\left(\phi^{\mathcal{H}}_{1:k} - \gamma\phi^{\mathcal{H}}_{2:k+1}\right)^{\dagger}$, where † indicates the pseudo-inverse. However, this solution is biased [3], since the independent variables $\phi^{\mathcal{H}}_t - \gamma\phi^{\mathcal{H}}_{t+1}$ are *noisy* samples of the expected difference $\mathbb{E}[\phi^{\mathcal{H}}(h) - \gamma\sum_{o\in O}\phi^{\mathcal{H}}(h\pi o)\Pr[h\pi o \mid h]]$. In other words, estimating the value function parameters w is an *error-in-variables* problem.

The *least squares temporal difference* (LSTD) algorithm finds a consistent estimate of w by right-multiplying the approximate Bellman equation (Equation 4) by ${\phi^{\mathcal{H}}_t}^{\mathsf{T}}$:

$$\hat{w}^{\mathsf{T}} = \tfrac{1}{k}\sum_{t=1}^{k}\mathcal{R}_t{\phi^{\mathcal{H}}_t}^{\mathsf{T}}\left(\tfrac{1}{k}\sum_{t=1}^{k}\phi^{\mathcal{H}}_t{\phi^{\mathcal{H}}_t}^{\mathsf{T}} - \tfrac{\gamma}{k}\sum_{t=1}^{k}\phi^{\mathcal{H}}_{t+1}{\phi^{\mathcal{H}}_t}^{\mathsf{T}}\right)^{-1} \tag{5}$$

Here, ${\phi^{\mathcal{H}}_t}^{\mathsf{T}}$ can be viewed as an *instrumental* variable [3], i.e., a measurement that is correlated with the true independent variables but uncorrelated with the noise in our estimates of these variables.

As the amount of data k increases, the empirical covariance matrices $\phi^{\mathcal{H}}_{1:k}{\phi^{\mathcal{H}}_{1:k}}^{\mathsf{T}}/k$ and $\phi^{\mathcal{H}}_{2:k+1}{\phi^{\mathcal{H}}_{1:k}}^{\mathsf{T}}/k$ converge with probability 1 to their population values, and so our estimate of the matrix to be inverted in (5) is consistent. So, as long as this matrix is nonsingular, our estimate of the inverse is also consistent, and our estimate of w converges to the true value with probability 1.

3 Predictive Features

LSTD provides a consistent estimate of the value function parameters w; but in practice, if the number of features is large relative to the number of training samples, then the LSTD estimate of w is prone to overfitting. This problem can be alleviated by choosing a small set of features that only contains information that is relevant for value function approximation. However, with the exception of LARS-TD [9], there has been little work on how to select features automatically for value function approximation when the system model is unknown; and of course, manual feature selection depends on not-always-available expert guidance. We approach the problem of finding a good set of features from a *bottleneck* perspective. That is, given a large set of features of history, we would like to find a compression that preserves only relevant information for predicting the value function J^{π}. As we will see in Section 4, this improvement is directly related to spectral identification of PSRs.

3.1 Finding Predictive Features Through a Bottleneck

In order to find a *predictive* feature compression, we first need to determine what we would like to predict. The most relevant prediction is the value function itself; so, we could simply try to predict total future discounted reward. Unfortunately, total discounted reward has high variance, so unless we have a lot of data, learning will be difficult. We can reduce variance by including other prediction tasks as well. For example, predicting *individual* rewards at future time steps seems highly relevant, and gives us much more immediate feedback. Similarly, future *observations* hopefully contain information about future reward, so trying to predict observations can help us predict reward.

We call these prediction tasks, collectively, *features of the future*. We write $\phi^{\mathcal{T}}_t$ for the vector of all features of the "future at time t," i.e., events starting at time $t+1$ and continuing forward. Now, instead of remembering a large *arbitrary* set of features of history, we want to find a small subspace of features of history that is relevant for predicting features of the future. We will call this subspace a *predictive compression*, and we will write the value function as a linear function of only the predictive compression of features. To find our predictive compression, we will use *reduced-rank regression* [10]. We define the following empirical covariance matrices between features of the future and features of histories:

$$\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} = \tfrac{1}{k}\sum_{t=1}^{k}\phi^{\mathcal{T}}_t{\phi^{\mathcal{H}}_t}^{\mathsf{T}} \qquad \widehat{\Sigma}_{\mathcal{H},\mathcal{H}} = \tfrac{1}{k}\sum_{t=1}^{k}\phi^{\mathcal{H}}_t{\phi^{\mathcal{H}}_t}^{\mathsf{T}} \tag{6}$$

Let $L_{\mathcal{H}}$ be the lower triangular Cholesky factor of $\widehat{\Sigma}_{\mathcal{H},\mathcal{H}}$. Then we can find a predictive compression of histories by a singular value decomposition (SVD) of the weighted covariance: write $\mathcal{U}\mathcal{D}\mathcal{V}^{\mathsf{T}} \approx \widehat{\Sigma}_{\mathcal{T},\mathcal{H}}L_{\mathcal{H}}^{-\mathsf{T}}$ for a truncated SVD [11], where $\mathcal{U}$ contains the left singular vectors, $\mathcal{V}$ contains the right singular vectors, and $\mathcal{D}$ is the diagonal matrix of singular values. (We can tune accuracy by keeping more or fewer singular values, i.e., columns of $\mathcal{U}$, $\mathcal{V}$, or $\mathcal{D}$.) We use the SVD to define a mapping $\widehat{U}$ from the compressed space up to the space of features of the future, and we define $\widehat{V}$ to be the optimal compression operator given $\widehat{U}$ (in a least-squares sense, see [12] for details):

$$\widehat{U} = \mathcal{U}\mathcal{D}^{1/2} \qquad \widehat{V} = \widehat{U}^{\mathsf{T}}\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}(\widehat{\Sigma}_{\mathcal{H},\mathcal{H}})^{-1} \tag{7}$$

By weighting different features of the future differently, we can change the approximate compression in interesting ways. For example, as we will see in Section 4.2, scaling up future reward by a constant factor results in a *value-directed compression*—but, unlike previous ways to find value-directed compressions [8], we do not need to know a model of our system ahead of time. For another example, let $L_{\mathcal{T}}$ be the Cholesky factor of the empirical covariance of future features $\widehat{\Sigma}_{\mathcal{T},\mathcal{T}}$. Then, if we scale features of the future by $L_{\mathcal{T}}^{-\mathsf{T}}$, the SVD will preserve the largest possible amount of mutual information between history and future, yielding a canonical correlation analysis [13, 14].

3.2 Predictive State Temporal Difference Learning

Now that we have found a predictive compression operator $\widehat{V}$ via Equation 7, we can replace the features of history $\phi_t^{\mathcal{H}}$ with the compressed features $\widehat{V}\phi_t^{\mathcal{H}}$ in the Bellman recursion, Equation 4:

$$w^{\mathsf{T}}\widehat{V}\phi_{1:k}^{\mathcal{H}} \approx \mathcal{R}_{1:k} + \gamma w^{\mathsf{T}}\widehat{V}\phi_{2:k+1}^{\mathcal{H}} \tag{8}$$

The least squares solution for w is still prone to an error-in-variables problem. The instrumental variable $\phi^{\mathcal{H}}$ is still correlated with the true independent variables and uncorrelated with noise, and so we can again use it to unbias the estimate of w. Define the additional covariance matrices:

$$\widehat{\Sigma}_{\mathcal{R},\mathcal{H}} = \tfrac{1}{k}\textstyle\sum_{t=1}^{k}\mathcal{R}_t\phi_t^{\mathcal{H}^{\mathsf{T}}} \qquad \widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}} = \tfrac{1}{k}\textstyle\sum_{t=1}^{k}\phi_{t+1}^{\mathcal{H}}\phi_t^{\mathcal{H}^{\mathsf{T}}} \tag{9}$$

Then, the corrected Bellman equation is $w^{\mathsf{T}}\widehat{V}\widehat{\Sigma}_{\mathcal{H},\mathcal{H}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} + \gamma w^{\mathsf{T}}\widehat{V}\widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}}$, and solving for w gives us the Predictive State Temporal Difference (PSTD) learning algorithm:

$$w^{\mathsf{T}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}}\left(\widehat{V}\widehat{\Sigma}_{\mathcal{H},\mathcal{H}} - \gamma\widehat{V}\widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}}\right)^{\dagger} \tag{10}$$

So far we have provided some intuition for why predictive features should be better than arbitrary features for temporal difference learning. Below we will show an additional benefit: the model-free algorithm in Equation 10 is, under some circumstances, equivalent to a model-based method which uses subspace identification to learn Predictive State Representations [6, 7].

4 Predictive State Representations

A predictive state representation (PSR) [15] is a compact and complete description of a dynamical system. Unlike POMDPs, which represent state as a distribution over a latent variable, PSRs represent state as a set of *predictions* of *tests*. Just as a history is an ordered sequence of action-observation pairs executed prior to time t, we define a *test* of length i to be an ordered sequence of action-observation pairs $\tau = a_1o_1\ldots a_io_i$ that can be executed and observed *after* time t [15]. The *prediction* for a test τ after a history h, written $\tau(h)$, is the probability that we will see the test observations $\tau^O = o_1\ldots o_i$, given that we *intervene* [16] to execute the test actions $\tau^A = a_1\ldots a_i$: $\tau(h) = \Pr[\tau^O \mid h, \mathrm{do}(\tau^A)]$. If $Q = \{\tau_1, \ldots, \tau_n\}$ is a set of tests, we write $Q(h) = (\tau_1(h), \ldots, \tau_n(h))^{\mathsf{T}}$ for the corresponding vector of test predictions.

Formally, a PSR consists of five elements $\langle A, O, Q, s_1, F\rangle$. A is a finite set of possible actions, and O is a finite set of possible observations. Q is a *core* set of tests, i.e., a set whose vector of predictions $Q(h)$ is a sufficient statistic for predicting the success probabilities of *all* tests. F is the set of functions f_τ which embody these predictions: $\tau(h) = f_\tau(Q(h))$. And, $m_1 = Q(\epsilon)$ is the initial prediction vector. In this work we will restrict ourselves to *linear* PSRs, in which all prediction functions are linear: $f_\tau(Q(h)) = r_\tau^{\mathsf{T}}Q(h)$ for some vector $r_\tau \in \mathbb{R}^{|Q|}$. Finally, a core set Q is *minimal* if the tests in Q are linearly independent [17, 18], i.e., no one test's prediction is a linear function of the other tests' predictions.

Since $Q(h)$ is a sufficient statistic for all tests, it is a *state* for our PSR: i.e., we can remember just $Q(h)$ instead of h itself. After action a and observation o, we can update $Q(h)$ recursively: if we write M_{ao} for the matrix with rows $r_{ao\tau}^{\mathsf{T}}$ for $\tau \in Q$, then we can use Bayes' Rule to show:

$$Q(hao) = \frac{M_{ao}Q(h)}{\Pr[o \mid h, \mathrm{do}(a)]} = \frac{M_{ao}Q(h)}{m_\infty^{\mathsf{T}}M_{ao}Q(h)} \tag{11}$$

where m_∞ is a normalizer, defined by $m_\infty^\top Q(h) = 1$ for all h. In addition to the above PSR parameters, for reinforcement learning we need a reward function $\mathcal{R}(h) = \eta^\top Q(h)$ mapping predictive states to immediate rewards, a discount factor $\gamma \in [0,1]$ which weights the importance of future rewards vs. present ones, and a policy $\pi(Q(h))$ mapping from predictive states to actions.

Instead of ordinary PSRs, we will work with *transformed PSRs* (TPSRs) [6, 7]. TPSRs are a generalization of regular PSRs: a TPSR maintains a small number of sufficient statistics which are *linear combinations* of a (potentially very large) set of test probabilities. That is, a TPSR maintains a small number of *feature predictions* instead of test predictions. TPSRs have exactly the same predictive abilities as regular PSRs, but are invariant under similarity transforms: given an invertible matrix S, we can transform $m_1 \to Sm_1$, $m_\infty^\top \to m_\infty^\top S^{-1}$, and $M_{ao} \to SM_{ao}S^{-1}$ without changing the corresponding dynamical system, since pairs $S^{-1}S$ cancel in Eq. 11. The main benefit of TPSRs over regular PSRs is that, given *any* core set of tests, low dimensional parameters can be found using spectral matrix decomposition and regression instead of combinatorial search. In this respect, TPSRs are closely related to the transformed representations of LDSs and HMMs found by *subspace identification* [19, 20, 14, 21].

4.1 Learning Transformed PSRs

Let Q be a minimal core set of tests, so that $n = |Q|$ is the linear dimension of the system. Then, let $\mathcal{T}$ be a larger core set of tests (not necessarily minimal), and let $\mathcal{H}$ be the set of all possible histories. As before, write $\phi_t^{\mathcal{H}} \in \mathbb{R}^\ell$ for a vector of features of history at time t, and write $\phi_t^{\mathcal{T}} \in \mathbb{R}^\ell$ for a vector of features of the future at time t. Since $\mathcal{T}$ is a core set of tests, by definition we can compute any test prediction $\tau(h)$ as a linear function of $\mathcal{T}(h)$. And, since feature predictions are linear combinations of test predictions, we can also compute any feature prediction $\phi(h)$ as a linear function of $\mathcal{T}(h)$. We define the matrix $\Phi^{\mathcal{T}} \in \mathbb{R}^{\ell \times |\mathcal{T}|}$ to embody our predictions of future features: an entry of $\Phi^{\mathcal{T}}$ is the weight of one of the tests in $\mathcal{T}$ for calculating the prediction of one of the features in $\phi^{\mathcal{T}}$. Below we define several covariance matrices, Equation 12(a–d), in terms of the observable quantities $\phi_t^{\mathcal{T}}$, $\phi_t^{\mathcal{H}}$, a_t, and o_t, and show how these matrices relate to the parameters of the underlying PSR. These relationships then lead to our learning algorithm, Eq. 14 below.

First we define $\Sigma_{\mathcal{H},\mathcal{H}}$, the covariance matrix of features of histories, as $\mathbb{E}[\phi_t^{\mathcal{H}} \phi_t^{\mathcal{H}\top} \mid h_t \sim \omega]$. Given k samples, we can approximate this covariance:

$$\widehat{\Sigma}_{\mathcal{H},\mathcal{H}} = \tfrac{1}{k} \phi_{1:k}^{\mathcal{H}} \phi_{1:k}^{\mathcal{H}\top}. \tag{12a}$$

As $k \to \infty$, $\widehat{\Sigma}_{\mathcal{H},\mathcal{H}}$ converges to the true covariance $\Sigma_{\mathcal{H},\mathcal{H}}$ with probability 1. Next we define $\Sigma_{\mathcal{S},\mathcal{H}}$, the cross covariance of states and features of histories. Writing $s_t = Q(h_t)$ for the (unobserved) state at time t, let $\Sigma_{\mathcal{S},\mathcal{H}} = \mathbb{E}\left[\tfrac{1}{k} s_{1:k} \phi_{1:k}^{\mathcal{H}\top} \,\middle|\, h_t \sim \omega\ (\forall t)\right]$. We cannot directly estimate $\Sigma_{\mathcal{S},\mathcal{H}}$ from data, but this matrix will appear as a factor in several of the matrices that we define below. Next we define $\Sigma_{\mathcal{T},\mathcal{H}}$, the cross covariance matrix of the features of tests and histories (see [12] for derivations):

$$\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} \equiv \tfrac{1}{k} \phi_{1:k}^{\mathcal{T}} \phi_{1:k}^{\mathcal{H}\top} \qquad \Sigma_{\mathcal{T},\mathcal{H}} \equiv \mathbb{E}[\phi_t^{\mathcal{T}} \phi_t^{\mathcal{H}\top} \mid h_t \sim \omega, \text{do}(\zeta)] = \Phi^{\mathcal{T}} R \Sigma_{\mathcal{S},\mathcal{H}} \tag{12b}$$

where row τ of the matrix R is r_τ, the linear function that specifies the prediction of the test τ given the predictions of tests in the core set Q. By do(ζ), we mean to approximate the effect of executing all sequences of actions required by all tests or features of the future at once. This is not difficult in our experiments (in which all tests use compatible action sequences); but see [12] for further discussion. Eq. 12b tells us that, because of our assumptions about linear dimension, the matrix $\Sigma_{\mathcal{T},\mathcal{H}}$ has factors $R \in \mathbb{R}^{|\mathcal{T}| \times n}$ and $\Sigma_{\mathcal{S},\mathcal{H}} \in \mathbb{R}^{n \times \ell}$. Therefore, the *rank* of $\Sigma_{\mathcal{T},\mathcal{H}}$ is no more than n, the linear dimension of the system. We can also see that, since the size of $\Sigma_{\mathcal{T},\mathcal{H}}$ is fixed, as the number of samples k increases, $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} \to \Sigma_{\mathcal{T},\mathcal{H}}$ with probability 1.

Next we define $\Sigma_{\mathcal{H},ao,\mathcal{H}}$, a *set* of matrices, one for each action-observation pair, that represent the covariance between features of history before and after taking action a and observing o. In the following, $\mathbb{I}_t(o)$ is an indicator variable for whether we see observation o at step t.

$$\widehat{\Sigma}_{\mathcal{H},ao,\mathcal{H}} \equiv \tfrac{1}{k} \textstyle\sum_{t=1}^{k} \phi_{t+1}^{\mathcal{H}} \mathbb{I}_t(o) \phi_t^{\mathcal{H}\top} \qquad \Sigma_{\mathcal{H},ao,\mathcal{H}} \equiv \mathbb{E}\left[\Sigma_{\mathcal{H},ao,\mathcal{H}} \middle| h_t \sim \omega\ (\forall t),\ \text{do}(a)\ (\forall t)\right] \tag{12c}$$

Since the dimensions of each $\widehat{\Sigma}_{\mathcal{H},ao,\mathcal{H}}$ are fixed, as $k \to \infty$ these empirical covariances converge to the true covariances $\Sigma_{\mathcal{H},ao,\mathcal{H}}$ with probability 1. Finally we define $\Sigma_{\mathcal{R},\mathcal{H}}$, and approximate the covariance (in this case a vector) of reward and features of history:

$$\widehat{\Sigma}_{\mathcal{R},\mathcal{H}} \equiv \tfrac{1}{k}\textstyle\sum_{t=1}^{k} \mathcal{R}_t \phi_t^{\mathcal{H}\mathsf{T}} \qquad \Sigma_{\mathcal{R},\mathcal{H}} \equiv \mathbb{E}[\mathcal{R}_t \phi_t^{\mathcal{H}\mathsf{T}} \mid h_t \sim \omega] = \eta^{\mathsf{T}} \Sigma_{\mathcal{S},\mathcal{H}} \tag{12d}$$

Again, as $k \to \infty$, $\widehat{\Sigma}_{\mathcal{R},\mathcal{H}}$ converges to $\Sigma_{\mathcal{R},\mathcal{H}}$ with probability 1.

We now wish to use the above-defined matrices to learn a TPSR from data. To do so we need to make a somewhat-restrictive assumption: we assume that our features of history are rich enough to determine the state of the system, i.e., the regression from $\phi^{\mathcal{H}}$ to s is exact: $s_t = \Sigma_{\mathcal{S},\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \phi_t^{\mathcal{H}}$. We discuss how to relax this assumption in [12]. We also need a matrix U such that $U^{\mathsf{T}} \Phi^{\mathcal{T}} R$ is invertible; with probability 1 a random matrix satisfies this condition, but as we will see below, there are reasons to choose U via SVD of a scaled version of $\Sigma_{\mathcal{T},\mathcal{H}}$ as described in Sec. 3.1. Using our assumptions we can show a useful identity for $\Sigma_{\mathcal{H},ao,\mathcal{H}}$ (for proof details see [12]):

$$\Sigma_{\mathcal{S},\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{\mathcal{H},ao,\mathcal{H}} = M_{ao} \Sigma_{\mathcal{S},\mathcal{H}} \tag{13}$$

This identity is at the heart of our learning algorithm: it states that $\Sigma_{\mathcal{H},ao,\mathcal{H}}$ contains a hidden copy of M_{ao}, the main TPSR parameter that we need to learn. We would like to recover M_{ao} via Eq. 13, $M_{ao} = \Sigma_{\mathcal{S},\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{\mathcal{H},ao,\mathcal{H}} \Sigma_{\mathcal{S},\mathcal{H}}^{\dagger}$; but of course we do not know $\Sigma_{\mathcal{S},\mathcal{H}}$. Fortunately, it turns out that we can use $U^{\mathsf{T}} \Sigma_{\mathcal{T},\mathcal{H}}$ as a stand-in, since this matrix differs only by an invertible transform (Eq. 12b).

We now show how to recover a TPSR from the matrices $\Sigma_{\mathcal{T},\mathcal{H}}$, $\Sigma_{\mathcal{H},\mathcal{H}}$, $\Sigma_{\mathcal{R},\mathcal{H}}$, $\Sigma_{\mathcal{H},ao,\mathcal{H}}$, and U. Since a TPSR's predictions are invariant to a similarity transform of its parameters, our algorithm only recovers the TPSR parameters to within a similarity transform [7, 12].

$$b_t \equiv U^{\mathsf{T}} \Sigma_{\mathcal{T},\mathcal{H}} (\Sigma_{\mathcal{H},\mathcal{H}})^{-1} \phi_t^{\mathcal{H}} = (U^{\mathsf{T}} \Phi^{\mathcal{T}} R) s_t \tag{14a}$$

$$B_{ao} \equiv U^{\mathsf{T}} \Sigma_{\mathcal{T},\mathcal{H}} (\Sigma_{\mathcal{H},\mathcal{H}})^{-1} \Sigma_{\mathcal{H},ao,\mathcal{H}} (U^{\mathsf{T}} \Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} = (U^{\mathsf{T}} \Phi^{\mathcal{T}} R) M_{ao} (U^{\mathsf{T}} \Phi^{\mathcal{T}} R)^{-1} \tag{14b}$$

$$b_\eta^{\mathsf{T}} \equiv \Sigma_{\mathcal{R},\mathcal{H}} (U^{\mathsf{T}} \Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} = \eta^{\mathsf{T}} (U^{\mathsf{T}} \Phi^{\mathcal{T}} R)^{-1} \tag{14c}$$

Our PSR learning algorithm is simple: replace each true covariance matrix in Eq. 14 by its empirical estimate. Since the empirical estimates converge to their true values with probability 1 as the sample size increases, our learning algorithm is clearly statistically consistent.

4.2 Predictive State Temporal Difference Learning (Revisited)

Finally, we are ready to show that the model-free PSTD learning algorithm introduced in Section 3.2 is *equivalent* to a model-based algorithm built around PSR learning. For a fixed policy π, a PSR or TPSR's value function is a linear function of state, $V(s) = w^{\mathsf{T}} s$, and is the solution of the PSR Bellman equation [22]: for all s, $w^{\mathsf{T}} s = b_\eta^{\mathsf{T}} s + \gamma \sum_{o \in O} w^{\mathsf{T}} B_{\pi o} s$, or equivalently, $w^{\mathsf{T}} = b_\eta^{\mathsf{T}} + \gamma \sum_{o \in O} w^{\mathsf{T}} B_{\pi o}$. Substituting in our learned PSR parameters from Equations 14(a–c), we get

$$w^{\mathsf{T}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} (U^{\mathsf{T}} \widehat{\Sigma}_{\mathcal{T},\mathcal{H}})^{\dagger} + \gamma \textstyle\sum_{o \in O} w^{\mathsf{T}} U^{\mathsf{T}} \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} (\widehat{\Sigma}_{\mathcal{H},\mathcal{H}})^{-1} \widehat{\Sigma}_{\mathcal{H},\pi o,\mathcal{H}} (U^{\mathsf{T}} \widehat{\Sigma}_{\mathcal{T},\mathcal{H}})^{\dagger}$$

$$w^{\mathsf{T}} U^{\mathsf{T}} \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} + \gamma w^{\mathsf{T}} U^{\mathsf{T}} \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} (\widehat{\Sigma}_{\mathcal{H},\mathcal{H}})^{-1} \widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}}$$

since, by comparing Eqs. 12c and 9, we can see that $\sum_{o \in O} \widehat{\Sigma}_{\mathcal{H},\pi o,\mathcal{H}} = \widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}}$. Now, define $\widehat{U}$ and $\widehat{V}$ as in Eq. 7, and let $U = \widehat{U}$ as suggested above in Sec. 4.1. Then $U^{\mathsf{T}} \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} = \widehat{V} \widehat{\Sigma}_{\mathcal{H},\mathcal{H}}$, and

$$w^{\mathsf{T}} \widehat{V} \widehat{\Sigma}_{\mathcal{H},\mathcal{H}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} + \gamma w^{\mathsf{T}} \widehat{V} \widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}} \implies w^{\mathsf{T}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} \left(\widehat{V} \widehat{\Sigma}_{\mathcal{H},\mathcal{H}} - \gamma \widehat{V} \widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}} \right)^{\dagger} \tag{15}$$

Eq. 15 is exactly Eq. 10, the PSTD algorithm. So, we have shown that, if we learn a PSR by the subspace identification algorithm of Sec. 4.1 and then compute its value function via the Bellman equation, we get the exact same answer as if we had directly learned the value function via the model-free PSTD method. In addition to adding to our understanding of both methods, an important corollary of this result is that PSTD is a *statistically consistent* algorithm for PSR value function approximation—to our knowledge, the first such result for a TD method.

5 Experimental Results

5.1 Estimating the Value Function of a RR-POMDP

We evaluate the PSTD learning algorithm on a synthetic example derived from [23]. The problem is to find the value function of a policy in a partially observable Markov decision Process (POMDP). The POMDP has 4 latent states, but the policy's transition matrix is low rank: the resulting belief distributions lie in a 3-dimensional subspace of the original belief simplex (see [12] for details).

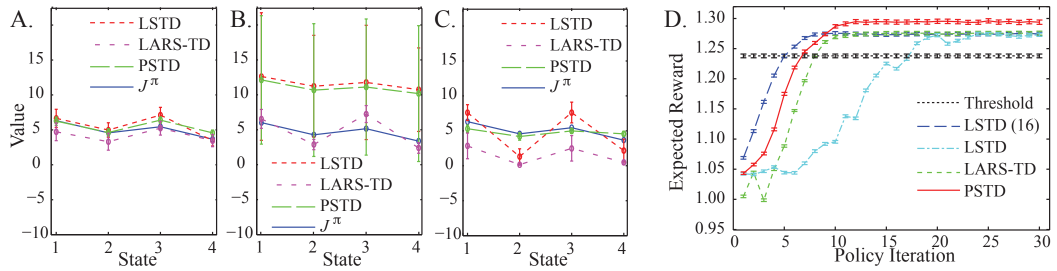

Figure 1: Experimental Results. Error bars indicate standard error. (A) Estimating the value function with a small number of informative features. All three approaches do well. (B) Estimating the value function with a small set of informative features and a large set of random features. LARS-TD is designed for this scenario and dramatically outperforms PSTD and LSTD. (C) Estimating the value function with a large set of semi-informative features. PSTD is able to determine a small set of compressed features that retain the maximal amount of information about the value function, outperforming LSTD and LARS-TD. (D) Pricing a high-dimensional derivative via policy iteration. The optimal threshold strategy (sell if price is above a threshold [24]) is in black, LSTD (16 canonical features) is in blue, LSTD (on the full 220 features) is cyan, LARS-TD (feature selection from set of 220) is in green, and PSTD (16 dimensions, compressing 220 features) is in red.

We perform 3 experiments, comparing the performance of LSTD, LARS-TD, and PSTD when different sets of features are used. In each case we compare the value function estimated by each algorithm to the true value function computed by $J^\pi = \mathcal{R}(I - \gamma T^\pi)^{-1}$. In the first experiment we execute the policy π for 1000 time steps. We split the data into overlapping histories and tests of length 5, and sample 10 of these histories and tests to serve as centers for Gaussian radial basis functions. We then evaluate each basis function at every remaining sample. Then, using these features, we learned the value function using LSTD, LARS-TD, and PSTD with linear dimension 3 (Figure 1(A)). Each method estimated a reasonable value function. For the second experiment, we added 490 random, uninformative features to the 10 good features and then attempted to learn the value function with each of the 3 algorithms (Figure 1(B)). In this case, LSTD and PSTD both had difficulty fitting the value function due to the large number of irrelevant features. LARS-TD, designed for precisely this scenario, was able to select the 10 relevant features and estimate the value function better by a substantial margin. For the third experiment, we increased the number of sampled features from 10 to 500. In this case, each feature was somewhat relevant, but the number of features was large compared to the amount of training data. This situation occurs frequently in practice: it is often easy to find a large number of features that are at least somewhat related to state. PSTD outperforms LSTD and LARS-TD by summarizing these features and *efficiently* estimating the value function (Figure 1(C)).

5.2 Pricing A High-dimensional Financial Derivative

Derivatives are financial contracts with payoffs linked to the future prices of basic assets such as stocks, bonds and commodities. In some derivatives the contract holder has no choices, but in more complex cases, the holder must make decisions, and the value of the contract depends on how the holder acts—e.g., with *early exercise* the holder can decide to terminate the contract at any time and receive payments based on prevailing market conditions, so deciding when to exercise is an optimal stopping problem. Stopping problems provide an ideal testbed for policy evaluation methods, since we can collect a single data set which lets us evaluate any policy: we just choose the "continue" action forever. (We can then evaluate the "stop" action easily in any of the resulting states.)

We consider the financial derivative introduced by Tsitsiklis and Van Roy [24]. The derivative generates payoffs that are contingent on the prices of a single stock. At the end of each day, the holder may opt to exercise. At exercise the holder receives a payoff equal to the current price of the stock divided by the price 100 days beforehand. We can think of this derivative as a "psychic call": the holder gets to decide whether s/he would like to have bought an ordinary 100-day European call option, at the then-current market price, 100 days ago. In our simulation (and unknown to the investor), the underlying stock price follows a geometric Brownian motion with volatility $\sigma = 0.02$ and continuously compounded short term growth rate $\rho = 0.0004$. Assuming stock prices fluctuate only on days when the market is open, these parameters correspond to an annual growth rate of $\sim 10\%$. In more detail, if w_t is a standard Brownian motion, then the stock price p_t evolves as $\nabla p_t = \rho p_t \nabla t + \sigma p_t \nabla w_t$, and we can summarize relevant state at the end of each day as a vector

$x_t \in \mathbb{R}^{100}$, with $x_t = (\frac{p_{t-99}}{p_{t-100}}, \frac{p_{t-98}}{p_{t-100}}, \ldots, \frac{p_t}{p_{t-100}})^{\mathsf{T}}$. This process is Markov and ergodic [24, 25]: x_t and x_{t+100} are independent and identically distributed. The immediate reward for exercising the option is $G(x) = x(100)$, and the immediate reward for continuing to hold the option is 0. The discount factor $\gamma = e^{-\rho}$ is determined by the growth rate; this corresponds to assuming that the risk-free interest rate is equal to the stock's growth rate, meaning that the investor gains nothing in expectation by holding the stock itself.

The value of the derivative, if the current state is x, is given by $V^*(x) = \sup_t \mathbb{E}[\gamma^t G(x_t) \mid x_0 = x]$. Our goal is to calculate an approximate value function $V(x) = w^{\mathsf{T}}\phi^{\mathcal{H}}(x)$, and then use this value function to generate a stopping time $\min\{t \mid G(x_t) \geq V(x_t)\}$. To do so, we sample a sequence of 1,000,000 states $x_t \in \mathbb{R}^{100}$ and calculate features $\phi^{\mathcal{H}}$ of each state. We then perform *policy iteration* on this sample, alternately estimating the value function under a given policy and then using this value function to define a new greedy policy "stop if $G(x_t) \geq w^{\mathsf{T}}\phi^{\mathcal{H}}(x_t)$."

Within the above strategy, we have two main choices: which features do we use, and how do we estimate the value function in terms of these features. For value function estimation, we used LSTD, LARS-TD, or PSTD. In each case we re-used our 1,000,000-state sample trajectory for all iterations: we start at the beginning and follow the trajectory as long as the policy chooses the "continue" action, with reward 0 at each step. When the policy executes the "stop" action, the reward is $G(x)$ and the next state's features are all 0; we then restart the policy 100 steps in the future, after the process has fully mixed. For feature selection, we are fortunate: previous researchers have hand-selected a "good" set of 16 features for this data set through repeated trial and error (see [12] and [24, 25]). We greatly expand this set of features, then use PSTD to synthesize a small set of high-quality combined features. Specifically, we add the entire 100-step state vector, the squares of the components of the state vector, and several additional nonlinear features, increasing the total number of features from 16 to 220. We use histories of length 1, tests of length 5, and (for comparison's sake) we choose a linear dimension of 16. Tests (but not histories) were value-directed by reducing the variance of all features *except reward* by a factor of 100.

Figure 1D shows results. We compared PSTD (reducing 220 to 16 features) to LSTD with either the 16 hand-selected features or the full 220 features, as well as to LARS-TD (220 features) and to a simple thresholding strategy [24]. In each case we evaluated the final policy on 10,000 new random trajectories. PSTD outperformed each of its competitors, improving on the next best approach, LARS-TD, by 1.75 percentage points. In fact, PSTD performs better than the best previously reported approach [24, 25] by 1.24 percentage points. These improvements correspond to appreciable fractions of the risk-free interest rate (which is about 4 percentage points over the 100 day window of the contract), and therefore to significant arbitrage opportunities: an investor who doesn't know the best strategy will consistently undervalue the security, allowing an informed investor to buy it for below its expected value.

6 Conclusion

In this paper, we attack the feature selection problem for temporal difference learning. Although well-known temporal difference algorithms such as LSTD can provide asymptotically unbiased estimates of value function parameters in linear architectures, they can have trouble in finite samples: if the number of features is large relative to the number of training samples, then they can have high variance in their value function estimates. For this reason, in real-world problems, a substantial amount of time is spent selecting a small set of features, often by trial and error [24, 25]. To remedy this problem, we present the PSTD algorithm, a new approach to feature selection for TD methods, which demonstrates how insights from system identification can benefit reinforcement learning. PSTD automatically chooses a small set of features that are relevant for prediction and value function approximation. It approaches feature selection from a *bottleneck* perspective, by finding a small set of features that preserves only *predictive* information. Because of the focus on predictive information, the PSTD approach is closely connected to PSRs: under appropriate assumptions, PSTD's compressed set of features is asymptotically equivalent to TPSR state, and PSTD is a consistent estimator of the PSR value function.

We demonstrate the merits of PSTD compared to two popular alternative algorithms, LARS-TD and LSTD, on a synthetic example, and argue that PSTD is most effective when approximating a value function from a large number of features, each of which contains at least a little information about state. Finally, we apply PSTD to a difficult optimal stopping problem, and demonstrate the practical utility of the algorithm by outperforming several alternative approaches and topping the best reported previous results.

References

[1] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[2] Justin A. Boyan. Least-squares temporal difference learning. In *Proc. Intl. Conf. Machine Learning*, pages 49–56. Morgan Kaufmann, San Francisco, CA, 1999.

[3] Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. In *Machine Learning*, pages 22–33, 1996.

[4] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, 2003.

[5] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 752–759, New York, NY, USA, 2008. ACM.

[6] Matthew Rosencrantz, Geoffrey J. Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *Proc. ICML*, 2004.

[7] Byron Boots, Sajid M. Siddiqi, and Geoffrey J. Gordon. Closing the learning-planning loop with predictive state representations. In *Proceedings of Robotics: Science and Systems VI*, 2010.

[8] Pascal Poupart and Craig Boutilier. Value-directed compression of pomdps. In *NIPS*, pages 1547–1554, 2002.

[9] J. Zico Kolter and Andrew Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 521–528, New York, NY, USA, 2009. ACM.

[10] Gregory C. Reinsel and Rajabather Palani Velu. *Multivariate Reduced-rank Regression: Theory and Applications*. Springer, 1998.

[11] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996.

[12] Byron Boots and Geoffrey J. Gordon. Predictive state temporal difference learning. Technical report, arXiv.org.

[13] Harold Hotelling. The most predictable criterion. *Journal of Educational Psychology*, 26:139–142, 1935.

[14] S. Soatto and A. Chiuso. Dynamic data factorization. Technical report, UCLA, 2001.

[15] Michael Littman, Richard Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[16] Judea Pearl. *Causality: models, reasoning, and inference*. Cambridge University Press, 2000.

[17] Herbert Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12:1371–1398, 2000.

[18] Satinder Singh, Michael James, and Matthew Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Proc. UAI*, 2004.

[19] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer, 1996.

[20] Tohru Katayama. *Subspace Methods for System Identification*. Springer-Verlag, 2005.

[21] Daniel Hsu, Sham Kakade, and Tong Zhang. A spectral algorithm for learning hidden Markov models. In *COLT*, 2009.

[22] Michael R. James, Ton Wessling, and Nikos A. Vlassis. Improving approximate value iteration using memories and predictive state representations. In *AAAI*, 2006.

[23] Sajid Siddiqi, Byron Boots, and Geoffrey J. Gordon. Reduced-rank hidden Markov models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-2010)*, 2010.

[24] John N. Tsitsiklis and Benjamin Van Roy. Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44:1840–1851, 1997.

[25] David Choi and Benjamin Roy. A generalized Kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynamic Systems*, 16(2):207–239, 2006.