On Discovery and Learning of Models with Predictive Representations of State for Agents with Continuous Actions and Observations

David Wingate
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109
wingated@umich.edu

Satinder Singh
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109
baveja@umich.edu

ABSTRACT

Models of agent-environment interaction that use predictive state representations (PSRs) have mainly focused on the case of discrete observations and actions. The theory of discrete PSRs uses an elegant construct called the system dynamics matrix and derives the notion of predictive state as a sufficient statistic via the rank of the matrix. With continuous observations and actions, such a matrix and its rank no longer exist. In this paper, we show how to define an analogous construct for the continuous case, called the system dynamics distributions, and use information theoretic notions to define a sufficient statistic and thus state. Given this new construct, we use kernel density estimation to learn approximate system dynamics distributions from data, and use information-theoretic tools to derive algorithms for discovery of state and learning of model parameters. We illustrate our new modeling method on two example problems.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

Predictive representations of state, information theory, dynamical system modeling

1. INTRODUCTION

Model building is often an important aspect of learning in autonomous agents. Many agent-environment interactions can be captured as dynamical systems, in which the agent repeatedly generates actions, the environment stochastically transitions between states, and the agent receives observations. There are many popular models for capturing such systems (POMDPs, for example), but in a reinforcement learning context, models that use predictive representations of state (or PSRs [6]) have recently attracted attention.

PSRs replace the traditional notion of state with a set of statistics about future actions and observations. A number of results have been obtained about PSRs. In the case of discrete observations, for example, PSRs have been shown to be just as accurate, expressive and compact as unstructured POMDPs. Because the state representation is grounded directly in data, PSRs can be learned directly from an agent's experience trajectories by a number of different algorithms [14, 7, 4]. They are also flexible: predictive representations have shown to be good bases for generalization [9], and have recently been generalized to be able to capture relational knowledge [17]. Most PSRs are only defined for discrete actions and observations; the only known exception is the Predictive Linear Gaussian model (or PLG) [11]. The PLG is not a direct extension of PSRs to the continuous case; it is derived more directly by transforming a linear dynamical system into a predictive form.

This paper extends PSRs more directly to the continuous case. To accomplish this, we introduce continuous analogues of the concepts central to PSRs, and modify all algorithms accordingly. Our extensions address two central problems in modeling partially observable dynamical systems: finding and updating sufficient statistics for history (that is, state). There are three central ideas which form the basis of our extensions. First, we allow all observations and actions to be real-valued. We therefore replace all probabilities with densities. Second, we define the *system dynamics distributions*. These distributions describe the evolution of the system over time, and are the continuous analogue of the system dynamics matrix used in PSRs. Third, in a discrete PSR, the system dynamics matrix is used to discover sufficient statistics. We use concepts from information theory to define sufficiency and to discover the needed statistics.

These conceptual extensions require companion algorithmic modifications. We use four key ideas: first, to estimate the system dynamics distributions, we use kernel density estimation. Second, to measure sufficiency, we use a generalized form of mutual information based on quadratic Renyi entropy. Third, to discover sufficient statistics, we use random sampling combined with entropy optimization. Fourth, we show how Nystrom approximations and homotopy optimization yield an efficient implementation. The final combination of ideas has appealing properties; in particular, there is a nice mathematical synergy between the elements.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

We conclude with experiments showing that continuous PSRs can be used to model agents in dynamical systems. We demonstrate the ideas on two example problems, one of which is a partially observable dynamical world consisting of an autonomous mobile robot. The agent has realistic perceptual and action models: camera images are observations, no a priori information about the effect of actions is given, and no automatic state is given. Empirically, the ideas are viable: we show reduced variance in one-step prediction accuracy and reduced MSE of one-step predictions.

2. DISCRETE AND CONTINUOUS PSRS

We begin by describing key conceptual ideas in discrete PSRs, and then our extensions to continuous PSRs.

2.1 Discrete PSRs

Here we assume the agent is embedded in a controlled, discrete time dynamical system with discrete actions and observations. At each time step i, the agent executes an action $a_i \in \mathcal{A}$ and receives an observation $o_i \in \mathcal{O}$.

Histories: A *history* is a sequence of alternating actions and observations $a_1 o_1 a_2 o_2 \cdots a_m o_m$ describing an agent's experience from the beginning of time through timestep m.

Tests: An *s-test* (or "sequence test," which we will also just call a "test"), denoted $a^1 o^1 a^2 o^2 \cdots a^n o^n$, describes a possible sequence of *future* actions and observations (note the distinction between superscripts for tests and subscripts for histories). A test *succeeds* if the observations of the test are obtained, given that the test's actions are taken. A *prediction* for a test $t = a^1 o^1 a^2 o^2 \cdots a^n o^n$ starting in history h is the probability that t will succeed when its actions are executed immediately following h. Formally, we define the prediction for a test from history h of length m to be $p(t|h) = \Pr(o_{m+1} = o^1, o_{m+2} = o^2, \cdots, o_{m+n} = o^n | h, a_{m+1} = a^1, \cdots, a_{m+n} = a^n)$. For ease of notation, we use the following shorthand: for a set of tests $T = \{t_1, t_2, \cdots t_n\}$, $p(T|h) = [p(t_1|h), p(t_2|h), \cdots p(t_n|h)]^T$ is a column vector of predictions.

The system dynamics vector: The *systems dynamics vector* [12] is a conceptual construct introduced to define PSRs. This vector describes the evolution of a dynamical system over time, and is representation-independent. Every possible test t has an entry in this vector, which represents $p(t|\emptyset)$ (that is, the prediction of t from the null history). Tests are arranged in length-lexicographic order, from shortest to longest.

The system dynamics matrix: The system dynamics matrix $\mathcal{D}$ is obtained by conditioning the system dynamics vector on all histories. In this matrix, the first row is the system dynamics vector (corresponding to the null history). Every possible history has a row in the matrix; the entries in that row are obtained by conditioning the system dynamics vector on that particular history. An entry in the matrix is the prediction of a particular test from a particular history:

$$\mathcal{D}_{ij} = p(t_j|h_i) = \frac{p(h_i t_j)}{p(h_i)}$$

Tests and histories are arranged length-lexicographically, with ever increasing test and history lengths. The matrix has an infinite number of rows and columns.

Sufficient statistics: The system dynamics matrix inherently defines a notion of sufficient statistic, whose size is equal to the rank of the matrix (shown to be finite for interesting cases, such as POMDPs [12]). For finite rank, there must be a set of core tests whose predictions correspond to linearly independent columns of $\mathcal{D}$. By definition, every other column can be computed as a weighted combination of these columns, so we say that the predictions of these core tests are *a linearly sufficient statistic* for the system.

State: The key idea of a PSR is to represent state as a set of *predictions about tests*, which represent possible future observations given possible future actions. A PSR maintains a set of *core* tests, the predictions of which are state in any history, and which allow it to make correct predictions about *any* test.

2.2 Continuous PSRs

We now conceptually extend PSRs to the case of continuous actions and observations. The first problem is that when moving to continuous states and actions, it is no longer possible to order all possible histories and tests, simply because both observations and actions are real-valued. This means that we cannot define the system dynamics matrix, and hence we cannot define sufficiency in terms of its rank. Here, we outline our alternative.

Histories and tests: These are defined in exactly the same way as for discrete PSRs, except that both actions and observations may be continuous and vector-valued. If observations are vectors in $\mathbb{R}^3$ and actions are vectors in $\mathbb{R}^2$, for example, then a length three history is a vector in $\mathbb{R}^{15}$.

The continuous system dynamics vector: We define this somewhat differently than the discrete system dynamics vector: each entry represents one timestep, and so the n'th entry contains $p(F^n|\emptyset)$, which is the full distribution representing densities of tests of length n, measured from the null history.

The system dynamics distributions: The system dynamics distributions are defined by conditioning the continuous system dynamics vector on histories of increasing length. There is one distribution for each combination of a history length and a future length. We say that $p(F^n|H^m) = p(H^m F^n)/p(H^m)$ is the density of a length n future from a length m history. These distributions play the same role as the system dynamics matrix: they give the density of any given future from any given history. We will often drop the superscripts n and m when no ambiguity results.

Sufficient statistics: The idea of using linear independence to define sufficiency is no longer applicable. To define a new concept of sufficiency, we turn to information theory. We start by treating history and the future as random variables, whose joint is described by the system dynamics distribution $p(HF)$. State is a function of history: given a particular history h_i, we will summarize that history into a state variable $s_i = f(h_i)$. It is well-known that no function of a random variable can increase the information between that variable and another variable. This is known as the *data processing inequality:*

$$I(X;Y) \geq I(X; f(Y))$$

with equality if and only if $f(Y)$ is sufficient for Y [1]. It is this fact that we will use as the basis for our measure of sufficiency: we will say a function f captures state if $I(F^n; H^m) = I(F^n; S = f(H^m))$ for all n, m. If f can be found such that equality is achieved, the resulting state has summarized all of the information about the past which is

Modeling

- Sample trajectories of the system; use the suffix-history method to slice into samples; each sample consists of a past (h_i) and future (f_i).
- Use density estimation to estimate $p(HF)$.
- To find the state for sample i, we condition $p(HF)$ on h_i, and use the resulting conditional distribution to evaluate $p(t_j|h_i)$ for each test t_j. This gives us a sample of S.
- Assume that temporally consecutive samples i and $i+1$ are available. We can generate paired samples s_i and s_{i+1}, as well as o_i and a_i. This allows us to generate transition models by constructing $p(S, A, O, S')$.

Discovery

- Randomly sample a set of tests T.
- Discover better tests by increasing information $I(F; S = f(H; \Theta))$; increase information by tuning parameters Θ, which are the actual test values in T.

State Updates

- Given $p(T|h)$, compute $p(T|hao)$ using either Bayesian inversion or conditional regression.

Figure 1: Outline of the continuous PSR strategy.

relevant for predicting the future.

State: There are many choices we could make for the parametric mapping f discussed previously; the above definition of sufficiency applies to all of them. We are interested in the particular class of PSRs. Given a history h_t, our state at time t will be an n vector, the j'th component of which represents the prediction of a specific test from h_t: $s_t^j = p(t_j|h_t)$. The set T of all t_j constitute our core tests.

3. LEARNING PSRS

Our goal is to learn a PSR directly from observed sequences of actions and observations. Given this data, there are two key problems. First, the discovery problem: which tests will be sufficient statistics? Second, the state update problem: given state for history h, how can we compute state for history hao? In order to solve the discovery and state update problem, our algorithms will need to solve an additional problem: that of estimating the system dynamics matrix/distributions from data.

3.1 Learning Discrete PSRs

In a discrete PSR, all three questions can be answered through the system dynamics matrix.

Discovery: The idea of linear sufficiency suggests procedures for discovering sufficient statistics: a set of core tests corresponds to a set of linearly independent columns of the system dynamics matrix. Existing discovery algorithms search for linearly independent columns, typically through repeated SVDs.

Updating state: Because core tests are sufficient statistics, they can be used to maintain state. In a PSR, for every s-test t, there is a weight vector $m_t \in \mathbb{R}^{|Q|}$ independent of history h such that the prediction $p(t|h) = m_t^T p(Q|h)$ for all h. Given a set of core tests Q, their predictions $p(Q|h)$, an action a and an observation o, the updated prediction for a core test $q_i \in Q$ is

$$p(q_i|hao) = \frac{p(aoq_i|h)}{p(ao|h)} = \frac{m_{aoq_i}^T p(Q|h)}{m_{ao}^T p(Q|h)} \ . \qquad (1)$$

This means that to maintain state, we only need to know m_{ao}, which are the weights for the *one-step tests*, and the m_{aoq_i}, which are the weights for the *one-step extensions*.

Estimating the system dynamics matrix: To estimate the system dynamics matrix, we estimate each entry using sample statistics. We use the suffix-history algorithm [18] to generate samples: given a trajectory of the system, we slice the trajectory into all possible histories and futures.

3.2 Learning Continuous PSRs

In some ways, the move to continuous observations and actions simplifies things: we can use function approximators, take gradients, etc. Here we broadly outline our strategy for learning continuous PSRs; Figure 1 summarizes the process.

Discovery: How can we find tests whose predictions are state? First, we introduce a parametric mapping $f(H; \Theta)$ from history to state. We can evaluate sufficiency by measuring the information between the future and state, which is a function of the past: $I(F; S = f(H; \Theta))$. To maximize sufficiency, we can manipulate the parameters Θ to maximize information. What are the parameters of the function f? They are the tests themselves—the questions that the tests are posing, or the ao values in the vector t_j. We discuss this in Section 5.

State and state updates: To represent state, we still use the prediction of a set of tests; the prediction of a test is now a *density*, instead of a probability. To update state, we may use the Bayesian inversion in Eq. 1, but there are other possibilities. We discuss this in Section 6.

Estimating the system dynamics distributions: The new modeling problem is to approximate the distributions $p(HF)$. To do this, we choose to use kernel density estimation; as discussed in Section 4, this makes it compatible with our information-theoretic ideas.

This completes the conceptual outline of our move to continuous PSRs. The next sections present more detail.

4. ESTIMATING THE SYSTEM DYNAMICS DISTRIBUTIONS

Modeling the system dynamics distributions is a density estimation problem. We choose to use kernel density estimation with a Gaussian kernel. To estimate $p(HF)$, we need samples of the joint distribution of history and future. To generate the needed samples, we use the suffix-history algorithm [18]. Given a long trajectory of actions and observations, we slice the trajectory into all possible combinations of history and future. For example, a length five trajectory is sliced into five samples: one for each of H^0F^5, H^1F^4, H^2F^3, H^3F^2, and H^4F^1.

Given a set of samples $x_1 \cdots x_N \in \mathbb{R}^d$ of the random variable X, our estimate of $p(X)$ is

$$p(X) = \frac{1}{N} \sum_{j=1}^{N} K(X, x_j; \sigma_j)$$

with a Gaussian kernel:

$$\begin{aligned} K(X, x_j; \sigma_j) &= G(X - x_j; \sigma_j) \\ &= \frac{1}{\sqrt{(2\pi\sigma_j)^d}} e^{-(X-x_j)^T(X-x_j)/2\sigma_j} \end{aligned}$$

where we have assumed the use of a spherical covariance matrix $\sigma_j I$, and where d is the dimension of the variable X. Because we are using spherical Gaussians, we can write similar expressions for joint densities:

$$p(X, Y) = \frac{1}{N} \sum_{j=1}^{N} G(X - x_j; \sigma_j^X) G(Y - y_j; \sigma_j^Y)$$

The choice of kernel density estimation is motivated for several reasons: it is compatible with our information theoretic measures, our gradient optimizer and our Nystrom approximations, as will be clear throughout the paper.

5. DISCOVERY

There are two elements to discovery: determining how many tests are needed, and determining which actions and observations should comprise the tests. Most discovery algorithms are concerned with discovering a *minimal* set of statistics. We do not address the issue of minimality; rather, we supply many more tests than needed, and focus on discovering the constituent elements.

Our approach is to start by sampling tests randomly from observed trajectories. We then improve those tests by computing the gradients of information with respect to the parameters of the tests, and performing steepest ascent in parameter space.

5.1 Generalized Information Measures

Our goal is to maximize the mutual information between state S and the future F:

$$I(F; S = f(H; \Theta)) = H(F) + H(S) - H(F, S) \ ,$$

but it is too difficult to optimize Shannon information directly. However, Kapur [5] has argued that if the aim is not to compute an exact value of information, but rather to extremize information, generalized measures of information may be used. These have the same maxima and minima, but may fit other design parameters better (for example, they may be computationally cheaper). We also wish to make as few assumptions about the form of the densities as possible.

Finding an easily optimizable, generalized information measure while making few assumptions about the density is a problem that has been dealt with by the entropy optimization community. The solution that has emerged in the literature has been to 1) use a generalized information measure based on generalized entropies, and 2) use a kernel density estimate with a Gaussian kernel [8][13][3]; this is why we chose to model our densities with Gaussian kernel density estimation.

By far, the most popular generalized entropy in use is Renyi's entropy, which is defined as:

$$H_{R_\alpha}(X) = \frac{1}{\alpha - 1} \log \int p(X)^\alpha dX$$

This measure generalizes Shannon's entropy, because in the limit as α approaches 1, Shannon's entropy is recovered. The choice of $\alpha = 2$ yields quadratic Renyi entropy:

$$H_{R_2}(X) = -\log \int p(X)^2 dX \tag{2}$$

which we will write as $H(X)$ when it is clear from context that Renyi entropy is intended. In conjunction with kernel density estimation and a Gaussian kernel, Eq. 2 can be evaluated in closed form, as explained next.

5.2 Identities and Derivatives Necessary

Here, we outline our particular choices of information measure, entropy measure and density estimate. As discussed previously, our objective is to maximize

$$I(F; S = f(H; \Theta)) = H(F) + H(S) - H(F, S)$$

by tuning the parameters Θ. We will adopt a steepest gradient ascent strategy[1]:

$$\Theta = \Theta + \eta \frac{\partial I}{\partial \Theta} = \Theta + \eta \sum_i \frac{\partial I}{\partial s_i} \frac{\partial s_i}{\partial \Theta} \tag{3}$$

where η is a learning rate. We can take derivatives of I with respect to the parameters directly, or we can invoke the chain rule and take derivatives with respect to state variables first (as shown in the right-hand side of Eq. 3). These are equivalent formulations of the derivatives.

There are two quantities we must compute: $\frac{\partial I}{\partial s_i}$ and $\frac{\partial s_i}{\partial \Theta}$.

5.2.1 Information change wrt. state samples

We begin by taking derivatives of information with respect to individual state samples:

$$\begin{aligned} \frac{\partial I}{\partial s_i} &= \frac{\partial H(F)}{\partial s_i} + \frac{\partial H(S)}{\partial s_i} - \frac{\partial H(F, S)}{\partial s_i} \\ &= \frac{\partial H(S)}{\partial s_i} - \frac{\partial H(F, S)}{\partial s_i} \end{aligned}$$

This will result in a vector describing which way sample i wishes to move in order to increase information. The entropy measure that we will use is quadratic Renyi entropy:

$$\begin{aligned} H(X) &= -\log \int p(X)^2 dx \\ &= -\log \int \left(\frac{1}{N} \sum_j G(X - x_j; \sigma_j^X) \right)^2 dX \\ &= -\log \frac{1}{N^2} \sum_{ij} G(x_i - x_j; \sigma_i^X + \sigma_j^X) \end{aligned}$$

where we have used the identity

$$\int G(X - x_i; \sigma_i) G(X - x_j; \sigma_j) dX = G(x_i - x_j; \sigma_i + \sigma_j).$$

Similarly, a joint density can be written as:

$$H(X, Y) = -\log \frac{1}{N^2} \sum_{ij} G(x_i - x_j; \sigma_i^X + \sigma_j^X) G(y_i - y_j; \sigma_i^Y + \sigma_j^Y).$$

The nice thing about these choices of entropy and density estimator is that we have a closed-form expression which has reduced to computing pairwise interactions between the data points used to construct the density. Note that there

[1] More sophisticated numerical methods using these gradients are also possible; in our experiments, steepest ascent outperformed them, and is simpler to explain.

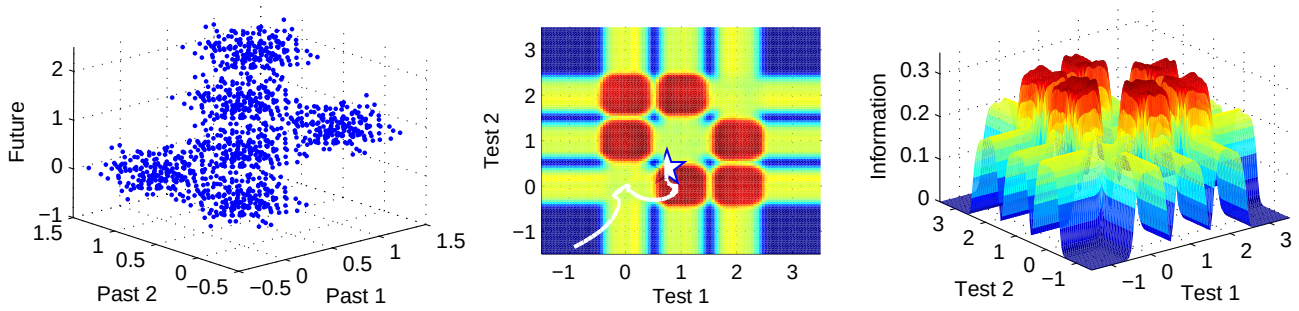

Figure 2: A toy data distribution (left), the resulting information landscape as a function of test value (right) and the trajectory taken by the gradient optimizer (center).

is no approximation in this integral, apart from the use of a kernel density estimate to begin with.

Another useful identity involves the derivative of a Gaussian:

$$\frac{\partial}{\partial x_i} G(x_i - x_j; \Sigma) = -G(x_i - x_j; \Sigma)(\Sigma^{-1})(x_i - x_j)$$

which we use to find the gradients of information:

$$\frac{\partial H(S)}{\partial s_i} = \frac{2}{H(S)} \sum_j G(s_i - s_j; \sigma_i^S + \sigma_j^S)(\sigma_i^S + \sigma_j^S)^{-1}(s_i - s_j)$$

and similarly

$$\frac{\partial H(F,S)}{\partial s_i} = \frac{2}{H(F,S)} \sum_j G(s_i - s_j; \sigma_i^S + \sigma_j^S)$$
$$G(f_i - f_j; \sigma_i^F + \sigma_j^F)(\sigma_i^S + \sigma_j^S)^{-1}(s_i - s_j)$$

5.2.2 *State sample changes wrt. parameters*

Finally, we need to compute the change in sample s_i with respect to the parameters Θ. Up to this point, none of the math that we have laid out says anything about predictive representations of state, and applies equally well to any parametric mapping from past to state. We will now introduce the choices that make this a PSR. Recall that $s_i = f(h_i; \Theta)$. s_i is a vector, the j'th component of which is the prediction of test t_j:

$$\begin{aligned} s_i^j &= p(t_j|h_i) = \frac{p(h_i, t_j)}{p(h_i)} \\ &= \frac{\frac{1}{N}\sum_k G(h_i - h_k; \sigma_k^H) G(t_j - f_k; \sigma_k^F)}{\frac{1}{N}\sum_l G(h_i - h_l; \sigma_l^H)} \\ &= \sum_k n(h_i)_k G(t_j - f_k; \sigma_k^F) \end{aligned}$$

where we have summarized the conditioning of the past into a function n (which only depends on h_i, and not on t_j).

As noted, the parameters of our f are the tests themselves. We can compute the partial of a given state variable with respect to the test values that generated it:

$$\frac{\partial s_i^j}{\partial t_j} = -\sum_k n(h_i)_k G(t_j - f_k; \sigma_k^F)(\sigma_k^F)^{-1}(t_j - f_k).$$

This completes the math that we require.

5.3 Example: A Toy Data Set

Here we consider a toy data set to illustrate the concepts of information and gradients, and to clarify exactly what the parameters are that we are trying to find. Consider one particular system dynamics distribution $p(H^2F^1)$ from an unspecified dynamical system. The system is uncontrolled, and observations are one-dimensional, so the distribution is three-dimensional, with two of the dimensions corresponding to "history" dimensions and one dimension corresponding to a "future" dimension. Samples from the distribution are shown in the leftmost panel of Figure 2 (we are only pretending that this data comes from a dynamical system; in reality, the data is from six Gaussian clusters in $\mathbb{R}^3$).

Suppose we decide to use two tests to summarize history. We will denote the prediction of test 1 as $p(F = l|H)$, and the prediction of test 2 as $p(F = k|H)$. Thus, our state is two-dimensional, and there are two parameters: l and k. Given particular values for l and k, we can compute the mutual information between F and S.

We wish to find the two best parameters to maximize $I(F|h; S = f(h))$. The rightmost panel of Figure 2 shows mutual information between state and the future as a function of the parameters. This information landscape highlights a few points of interest: values for l and k which are very far away from the high-density areas of the data (say, $l = -1, k = -1$) have low information content. It also shows that when l and k have the same value, no new information is added – in other words, the predictions are redundant.

The middle panel shows the results of the gradient optimization starting the tests at $[l = -1, k = -1.4]$, and ending at the star (it does not climb quite as one might expect because of the use of homotopy optimization, as discussed later). The gradient optimizer indeed moves the tests from regions of low information to regions of high information.

6. STATE UPDATING

Finally, we address the question of state updating. From a given history, we can estimate the densities of our core tests $p(T|h)$ using our estimates of the system dynamics distributions. With a new action and observation, we must update state to compute $p(T|hao)$. There are two possibilities.

Bayesian Inversion. By direct analogy to Eq. 1, we can perform the state update using Bayesian inversion:

$$p(t_j|hao) = \frac{p(aot_j|h)}{p(ao|h)}$$

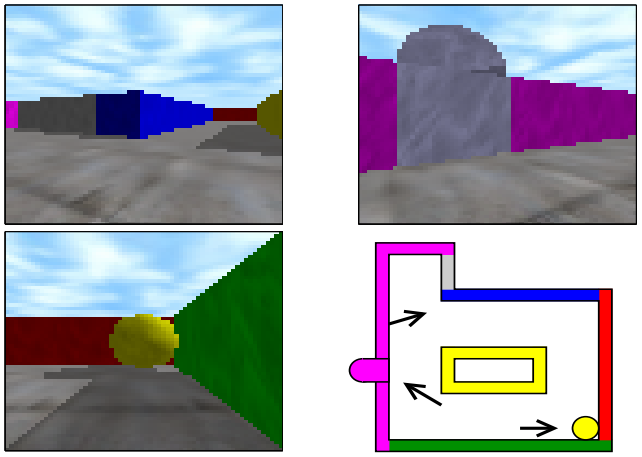

Figure 3: The autonomous robot domain. Arrows on the map correspond to viewpoints.

where we could use the system dynamics distributions to compute the needed densities. However, we prefer to find a recursive solution: we wish to update state in terms of previous state. Since state is sufficient for history, we can equally well model

$$p(t_j|hao) = \frac{p(aot_j|S_h)}{p(ao|S_h)} = \frac{p(S_h, a, o, t_j)}{p(S_h, a, o)}$$

using the same tools of density estimation:

$$\frac{p(S_h, a, o, t_j)}{p(S_h, a, o)} = \frac{\sum_j G(S_h - s_j)G(a - a_j)G(o - o_j)G(t_j - t_j)}{\sum_j G(S_h - s_j)G(a - a_j)G(o - o_j)}$$

where we assume that we have joint samples of state, action, observation, and future, as discussed in Fig. 1.

Conditional Regression. Another option is to adopt a function approximation view: we wish to compute $p(t_j|S_hao)$ as a black-box function of state, action, and observation: $p(t_j|S_hao) = f_{t_j}(S_ha, o,)$. Most function approximators attempt to approximate $\mathrm{E}[Y|X]$, which we can compute directly using the distributions (using S' to represent the new state following S, a, o):

$$\mathrm{E}\left[S'|S, a, o\right] = \frac{\sum_j G(S' - s'_j)G(a - a_j)G(o - o_j)G(S - S_j)}{\sum_j G(a - a_j)G(o - o_j)G(S - s_j)} \quad (4)$$

where we again assume we have samples of the needed joint densities, as discussed in Fig. 1.

7. EXPERIMENTS AND RESULTS

We have introduced several new ideas and made many design choices. Here, we present some experiments designed to explore the decisions we have made. We tested on two problems: a bouncing ball, and a simulated autonomous robot.

The Bouncing Ball. The first domain is an uncontrolled, nonlinear, two-dimensional dynamical system consisting of a ball bouncing. The ball bounces vertically, with a damped restitution when it strikes the floor. True hidden state is the position and velocity of the ball; only position is observed.

To model this system, we made an assumption: we use a suffix-history with a history of length 3 and a future of length 3 (we also model the one-step extended distribution with a history of length 3 and a future of length 4 for the Bayesian inversion state update). This is a choice based on intuition and experimentation (in general, more sophisticated methods of selecting which distributions to use are needed). Different experiments used different numbers of tests, as explained later.

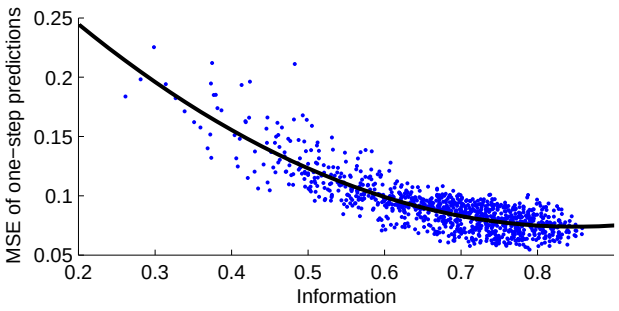

Figure 4: Information vs. one-step MSEs.

We trained on 2,000 data points. We initialized the state of the system to a random position and velocity, and ran the system for several timesteps; we then sliced the resulting trajectory into samples using suffix-history.

The Autonomous Robot Domain. The second domain is more challenging: a simulated autonomous mobile robot in a 2D maze. The domain is controlled, nonlinear, and partially observable; no a priori knowledge about the domain is given to the agent. The robot has two continuous actions (the amount by which to rotate and amount by which to move forward/backward), and continuous state coordinates (position x, y and orientation θ). The robot is located in a maze with obstacles and brightly colored walls. The observation is generated as follows: the agent's camera initially samples a 64x64 full color image, but the agent extracts a single feature from the image: the dominant color in the center of the image (done by convolving with a Gaussian). Observations are therefore three dimensional (consisting of rgb color values). With full camera images, about 80% of the states can be disambiguated through an observation, but with the reduction to a single color, the observability is severely reduced. Figure 3 shows representative camera images, as well as the map used. All actions are deterministic.

The training data is a single long trajectory of actions and observations (100,000 samples, generated with a movement policy that was a smoothed version of random exploration). Again, we assumed that length 3 histories and length 3 futures were sufficient. In this case, both history and future are 15-dimensional vectors (3 steps of history x (2 action dimensions + 3 observation dimensions)). The gradient optimizer used a subset of 10,000 samples; testing used the full 100,000 (either size data set is feasible with [and *only* with] the Nystrom approximations discussed in Section 7.2).

Error measure. We evaluate based on mean-squared error of the one-step predictions. That is, at each timestep, the agent is asked to predict the expected next observation. The true observation is given; there is some error, which we square. The mean is taken over the length of the test sequence. The absolute value of the MSE is not important; but rather the difference before and after application of the gradient optimizer.

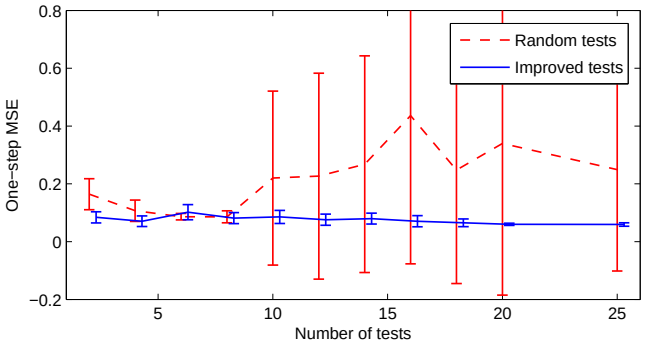

Figure 5: Improving tests on the ball domain.

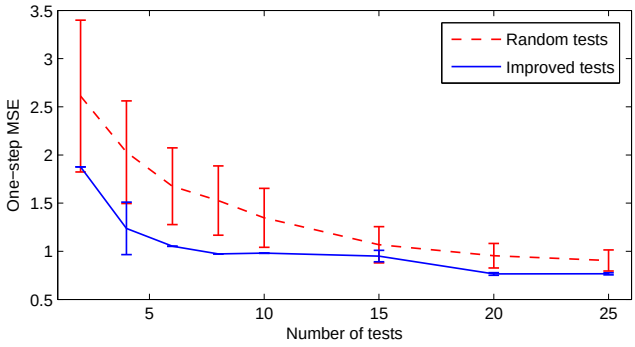

Figure 6: Improving tests on the robot domain.

Implementation. We used the optimizations discussed in Section 7.2. We used a Nystrom-based gradient optimizer with 100 landmarks. We used the same homotopy schedule ($\lambda = [500, 200, 100, 60, 20, 5, 1]$) for both problems, and set stepsizes such that the norm of the gradient vector was between 0.5 and 0.01 (depending on λ). All samples used the same covariances; this simplifies the math further.

7.1 Results

First, we directly test the hypothesis of the paper: that there is a correlation between higher information (between state and the future) and lower MSE of one-step predictions. To explore this, we sampled 1,000 random three-step tests in the ball domain. For each, we computed the MSE of using those tests to generate states, as well as the mutual information of states with the future. Figure 4 plots the resulting MSEs versus information (and is fit with a 3rd degree polynomial). There is an obvious correlation, which suggests that the idea of using Renyi entropy to solve the discovery problem may be feasible.

We next ask: does the information gradient optimizer work? We explored this question for both domains. For each domain, we fixed the number of core tests. We randomly sampled that many tests and computed the MSE of using them to generate states. We then optimized the tests with the gradient method, used the improved tests to generate states, and again computed the MSE. This was done for different numbers of core tests. Figure 5 shows results for the ball domain, while Figure 6 shows results for the vision domain. The number of tests used is the horizontal axis, while MSE of random and optimized tests is shown on the vertical axis.

The results are very encouraging. Both figures demonstrate the same behavior: not only does the optimizer consistently find tests which generate lower MSEs, it also reduces variance in the MSE. This is because the homotopy optimizer was able to consistently locate almost the same points, regardless of initial conditions. In the ball domain, the variance is particularly high for the randomly sampled tests. Sometimes, a bad set of tests can result in a catastrophic run of the system, resulting in very high MSE; the optimized tests never showed this behavior. Figure 7 illustrates this behavior. Note that the random tests, with few exceptions, never performed as well as the optimized tests. It is possible that this is because the optimized tests are not constrained to lie on the manifold of observed trajectories.

We conducted other experiments which are not reported in detail here. We noticed, for example, that there was generally not a difference between Bayesian inversion and conditional regression, except that the inversion method sometimes had higher variance.

7.2 Practical Considerations

We found that learning from large datasets was impossible without the addition of additional computational tools. We found the following techniques indispensable.

Homotopy optimization. As with all gradient methods, ours is guaranteed to only find a local maximum. However, the local maximum can be improved by "smoothing" the information landscape, and finding a local optimum of the smoothed landscape, and then gradually unsmoothing the landscape while continuing to optimize. This is known as homotopy optimization (or deformation optimization) because it uses a homotopy, which is a continuous transformation of an easy optimization problem into a hard one.

We accomplish this by smoothing test predictions: instead of computing $p(T|h) = \sum G(T - f_k|h, \sigma^T)$, we compute $p(T|h) = \sum G(T - f_k|h, \lambda\sigma^T)$. Using λ to scale the variance effectively makes test predictions look more similar, which has the side effect of smoothing the information landscape. By gradually reducing λ while taking gradient steps, a much better optimum is achieved.

Nystrom approximations. The complexity of the gradient computations is quadratic in the number of data points. One of the keys to scalability is the observation that the Gaussian used in our kernel density estimator is also a Mercer kernel. This means that Nystrom approximations can be used to simplify expressions of the form $E = \sum_{ij} G(x_i - x_j; \Sigma) = 1^T G 1$, where G is a matrix with $G_{ij} = G(x_i - x_j; \Sigma)$ and 1 is an appropriately sized column vector of ones.

Nystrom approximations work by selecting a number of "landmark" points: instead of computing a Gaussian of every point with every other point (G_{ij}), we compute the Gaussians with respect to a set of landmarks. Let $V_{ik} = G(l_i - x_k; \Sigma)$ be a matrix containing the kernel of every data point with each landmark l_i, and let $M_{kl} = G(l_k - l_l; \Sigma)$ be a matrix containing the kernel of every landmark with every other landmark. Then $E \approx (1^T V^T) M^{-1} (V 1)$, and we see that the expression $V^T M^{-1} V$ is effectively a low-rank approximation of the matrix G.

The complexity of this method is $O(d^3 + nd)$, where n

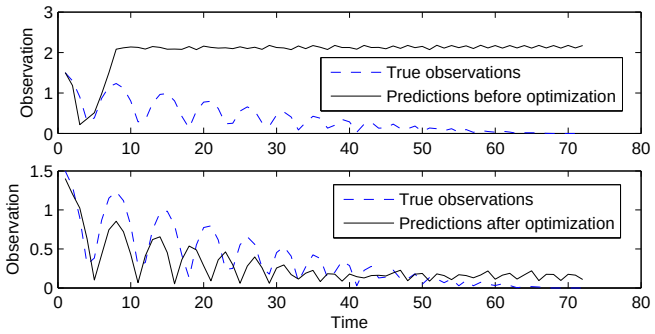

Figure 7: Top: a catastrophic run using random tests. Bottom: optimized tests avoid catastrophe.

is the total number of samples and $d \ll n$ is the number of landmark points. The complexity of the naive version is $O(n^2)$. Landmark points can be selected in a variety of ways; a good way is the dictionary-based methods of Engel [2], which ensure that the landmarks are linearly independent, and therefore that M is invertible.

8. RELATED WORK

The only other known extension of PSRs to the continuous case is the Predictive Linear-Gaussian model (or PLG) [11]. The PLG operates on a discrete time, linear dynamical system (LDS). An LDS is defined by a linear state update equation, and a companion observation process in which the observation is a (different) linear function of state. Both the transitions and observations can be corrupted with mean-zero Gaussian noise. These are the same assumptions used by the Kalman filter. The original PLG operated on linear, uncontrolled systems; the cPLG [10] operated on controlled linear systems, and the KPLG and MPLG operated on uncontrolled, nonlinear systems [15, 16]. However, no work has been done on the controlled, nonlinear case. This work fills that gap, but from a very different perspective.

9. CONCLUSIONS AND FUTURE WORK

We have extended PSRs to the continuous case with two core ideas: we have replaced the system dynamics matrix with the system dynamics distributions, and we have replaced the idea of using rank analysis to find sufficient statistics with ideas from information theory. We have argued that mutual information can help quantify the sufficiency of a candidate state representation; because information can be optimized, the representation can be improved.

We have also made several contributions on the algorithmic side, where there is a nice synergy between the elements: we started by using kernel density estimation to estimate the system dynamics distributions. Not only is it a nonparametric estimator, but it leads to closed-form expressions for mutual information. In addition, we can compute gradients of information with respect to test parameters in closed-form, which allows us to help solve the discovery problem. Both measuring information and computing gradients can be approximated efficiently; the resulting algorithms can handle tens or hundreds of thousands of data points.

Empirically, our ideas seem viable. Our continuous PSRs are a reasonable model of the dynamical systems presented, which includes an example of an agent embodied in a complex environment. There appears to be a correlation between mutual information and MSE which our optimization procedure exploits; experimentally, it reduces both the MSE of one-step predictions and the variance of the MSE.

There are still many open problems. Future work will address questions such as picking test and history dimensionality, and how to best exploit graphical structure in the system dynamics distributions.

Acknowledgments

David Wingate is supported under a National Science Foundation Graduate Research Fellowship. Satinder Singh is supported by NSF grant IIS-0413004. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

10. REFERENCES

[1] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[2] Y. Engel, S. Mannor, and R. Meir. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *ICML*, 2003.

[3] K. E. Hild, D. Erdogmus, and J. C. Principe. Blind source separation using renyi's mutual information. *IEEE Signal Processing Letters*, 8(6):174–176, 2001.

[4] M. R. James and S. Singh. Learning and discovery of predictive state representations in dynamical systems with reset. In *ICML*, pages 417–424, 2004.

[5] J. Kapur. *Measures of Information and their Application*. John Wiley, 1994.

[6] M. L. Littman, R. S. Sutton, and S. Singh. Predictive representations of state. In *NIPS*, pages 1555–1561, 2002.

[7] P. McCracken and M. Bowling. Online discovery and learning of predictive state representations. In *NIPS*, pages 875–882, 2006.

[8] J. C. Principe, D. Xu, and J. W. Fisher. Information theoretic learning. pages 265–319, 1999.

[9] E. J. Rafols, M. B. Ring, R. S. Sutton, and B. Tanner. Using predictive representations to improve generalization in reinforcement learning. In *IJCAI*, pages 835–840, 2005.

[10] M. Rudary and S. Singh. Predictive linear-Gaussian models of controlled stochastic dynamical systems. In *ICML*, 2006.

[11] M. Rudary, S. Singh, and D. Wingate. Predictive linear-Gaussian models of stochastic dynamical systems. In *UAI*, pages 501–508, 2005.

[12] S. Singh, M. R. James, and M. R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *UAI*, pages 512–519, 2004.

[13] K. Torkkola. Feature extraction by non-parametric mutual information maximization. *Journal of Machine Learning Research*, (3):1415–1438, 2003.

[14] E. Wiewiora. Learning predictive representations from a history. In *ICML*, pages 964–971, 2005.

[15] D. Wingate and S. Singh. Kernel predictive linear Gaussian models for nonlinear stochastic dynamical systems. In *ICML*, 2006.

[16] D. Wingate and S. Singh. Mixtures of predictive linear Gaussian models for nonlinear stochastic dynamical systems. In *AAAI*, 2006.

[17] D. Wingate, V. Soni, B. Wolfe, and S. Singh. Relational knowledge with predictive representations of state. In *IJCAI*, 2007.

[18] B. Wolfe, M. R. James, and S. Singh. Learning predictive state representations in dynamical systems without reset. In *ICML*, pages 980–987, 2005.