Learning Low Dimensional Predictive Representations

Matthew Rosencrantz MROSEN@CS.CMU.EDU
Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Geoff Gordon GGORDON@CS.CMU.EDU
CALD, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Sebastian Thrun THRUN@STANFORD.EDU
Stanford AI Laboratory, Stanford University, 353 Serra Mall, Stanford, CA 94305-9010, USA

Abstract

Predictive state representations (PSRs) have recently been proposed as an alternative to partially observable Markov decision processes (POMDPs) for representing the state of a dynamical system (Littman et al., 2001). We present a learning algorithm that learns a PSR from observational data. Our algorithm produces a variant of PSRs called transformed predictive state representations (TPSRs). We provide an efficient principal-components-based algorithm for learning a TPSR, and show that TPSRs can perform well in comparison to Hidden Markov Models learned with Baum-Welch in a real world robot tracking task for low dimensional representations and long prediction horizons.

1. Introduction

Predictive state representations (PSRs) have recently emerged as an alternative to partially observable Markov decision processes (POMDPs) for representing dynamical systems. Since they were first described by Littman et al. (2001), PSRs have attracted interest because they have been shown to have the same representational power as POMDPs, but are potentially more compact. POMDPs have been studied extensively in the literature (Cassandra et al., 1994; Cassandra et al., 1997) but planning in POMDPs poses difficulties since the algorithms are generally exponential in the state space of the model. PSRs have not yet been fully examined, and it is hoped that the potential compactness of the representation will lead to easier planning problems and more easily learned models. In this work we restrict ourselves to linear PSRs. The vast majority of the literature on PSRs relates to linear PSRs. (In fact we only know of a single paper that addresses non-linear PSRs, and then only for a specific deterministic case (Rudary & Singh, 2003).)

PSRs seek to represent a dynamical system by fixing a set of tests or queries about the world that could be executed and maintaining probability distributions over the success or failure of those tests. This is in contrast to POMDPs, which attempt to keep probability distributions directly over all world states. In Section 2 we will review the technical details of PSRs, but even from the above brief description the three main problems related to PSRs are apparent. First, there are an infinite number of possible queries in any realistic environment so it is infeasible to maintain probability distributions over all of them. Therefore, we must find a small set of queries that forms a sufficient statistic for the dynamical system. Singh et al. (2003) have called this the discovery problem. Second, once we have a small sufficient set of tests, we need to learn how to maintain a probability distribution over the success of these tests as the dynamical system progresses. We will refer to this second task as the learning problem. A third obvious problem is the planning problem, that is, given a distribution over the success of tests, decide what action we should take to maximize some notion of future reward. In this paper we address only the first two of these fundamental problems.

We propose a variant of PSRs called transformed predictive state representations or TPSRs. Instead of maintaining probability distributions over the outcomes of a set of tests, we will maintain linear combinations of these probabilities. We present a new and efficient PCA-based algorithm for learning the parameters of TPSRs and show how TPSRs help to alleviate the discovery problem. In addition, we present experimental results where we learn a TPSR to predict the outcome of a fixed policy in a real-world robot

Appearing in *Proceedings of the 21^{st} International Conference on Machine Learning*, Banff, Canada, 2004. Copyright 2004 by the first author.

navigation task. To our knowledge this is the first attempt to build a predictive state representation from real robot data. We compare TPSRs operating with fixed policies to Hidden Markov Models (HMMs) on our data (POMDPs with fixed policies are equivalent to HMMs). Through this comparison we demonstrate that TPSRs have important advantages over HMMs, and perform competitively with HMMs learned via the Baum-Welch algorithm. We show that TPSRs perform particularly well when the dimensionality of their model is restricted.

2. Predictive Representations

In this section we briefly review predictive state representations as described by Littman et al. (2001). A predictive state representation (PSR) is a compact and complete description of a dynamical system. PSRs represent their belief about the state of the world as a set of probability distributions over tests. Tests are a sequence of actions and observations that can be executed at a given time. A test is executed if we execute each of its specified actions in order; a test succeeds if it is executed and the observations produced by the dynamical system match those specified in the test. For example, if a dynamical system has actions $\{1, 2, 3\}$ and observations $\{a, b, c\}$, then a run of the system might produce the action-observation sequence $1b2a1c1c3a3b3c$ starting at time 0. Given this sequence we could say that the test $2a1c1c$ was executed successfully at time 1, the test $1c1b3a$ was executed unsuccessfully at time 2, and that the test $3a2a$ was never executed. The idea behind a PSR is that, if we knew the expected outcomes of executing all possible tests, we would know everything there is to know about the state of the dynamical system. That is, PSR test probabilities form a statistic about the state of the dynamical system equivalent to a POMDP belief state.

2.1. Representation

Formally a PSR consists of five elements $\{A, O, Q, s_0, F\}$. A is the set of actions that can be executed at each time-step and O is the set of possible observations. Q is a set of tests that constitute a sufficient statistic of the dynamical system. A sufficient set Q has the property that, for *any* test q, there exists some function f_q such that $p(q \mid h) = f_q(p(Q \mid h))$ for all histories h. Here

$$p(Q \mid h) = [\, p(q_1 \mid h), \ldots, p(q_K \mid h)\,]$$

is the row vector which contains the probabilities of success of the tests in Q, and $K = |Q|$. Note that $p(q \mid h)$ is the probability of q succeeding *if we execute it*, that is, it is conditioned on actually carrying out the actions specified in q. So long as we know the probabilities for the tests in Q we can compute the probabilities for all other tests; so, by analogy with POMDPs, we call the vector of probabilities $P(Q \mid h)$ a *belief state* or a *belief.* The vector s_0 is the initial belief over the outcomes of the tests in Q; if ϕ is the empty history then $s_0 = p(Q \mid \phi)$. Finally, F is a particular set of functions f_q that we will need to know in order to update our state when we take actions and receive new observations; the exact contents of this set will be described in Section 2.2.

As mentioned above, a probability distribution over the outcomes of all possible tests completely describes our belief about the state of the dynamical system, and since the set Q forms a sufficient statistic for the system, we need only maintain a distribution over the outcomes of the tests in Q. An important property of PSRs is that they are at least as compact as POMDPs: the number of test in a minimal PSR is less than or equal to the number of states in minimal POMDP. This inequality may be strict: Jaeger (2000) presents a "probability clock" example that linear PSRs can model infinitely more compactly than HMMs. To date no learning procedure for finding very compact PSR models has been proposed, and this work is an effort to address that problem.

2.2. State Update

In order to maintain a distribution over the tests in Q we need to compute the distribution over the test outcomes given a new extended history $p(Q \mid hao)$ from the current distribution $p(Q \mid h)$. (Here hao is the history h extended by the action a and the observation o.) Using Bayes' Rule:

$$p(q_i \mid hao) = \frac{p(aoq_i \mid h)}{p(ao \mid h)} = \frac{f_{aoq_i}(p(Q \mid h))}{f_{ao}(p(Q \mid h))}$$

The functions f_{aoq_i} and f_{ao} are precisely the functions that we need in the set F mentioned in Section 2.1. These functions produce predictions for the one step extensions to the tests in Q.

As noted in the introduction, we restrict ourselves to linear PSRs. In linear PSRs the functions f_x are required to be linear in the belief vector $P(Q \mid h)$, so we have

$$f_x(P(Q \mid h)) = P(Q \mid h) l_x$$

for some vector l_x. In this case the update equation can be rewritten:

$$p(q_i \mid hao) = \frac{p(aoq_i \mid h)}{p(ao \mid h)} = \frac{p(Q \mid h)\, l_{aoq_i}}{p(Q \mid h)\, l_{ao}} \tag{1}$$

To express this update rule in matrix form we write r^t for the current belief vector $p(Q \mid h)$ and r^{t+1} for the succeeding belief vector $p(Q \mid hao)$. We also write L_{ao} for the $K \times K$ matrix whose columns are l_{aoq_i} for $q_i \in Q$. With this notation the update becomes:

$$r^{t+1} = \frac{r^t L_{ao}}{r^t l_{ao}} \tag{2}$$

Learning linear PSRs involves learning the rational function in equation 2, and specifically the matrices L_{ao} and the vectors l_{ao}. Choosing a sufficient set of tests Q is called the discovery problem. In this work we present an algorithm which learns PSR parameters, and which partially addresses the discovery problem by requiring only that we specify a large set of tests that contains a sufficient subset.

3. Transformed Predictive Representations

Transformed predictive state representations (TPSRs) are a variant of PSRs. TPSRs are capable of tuning the complexity of their representation to match the specific problem being addressed. Instead of maintaining probability distributions over the outcomes of a small set of tests, TPSRs maintain a small number of linear combinations of the probabilities of a larger number of tests. TPSRs include PSRs as a special case, since we can always pick a linear combination which only includes a single test. But, introducing this extra generality allows us to use a singular value decomposition (SVD) to tune the complexity of our representation, as described in Section 3.2. If we write x^t for the row vector which contains the values of the linear combinations at time t, then the update rule for a TPSR is similar to equation 2:

$$x^{t+1} = \frac{x^t M_{ao}}{x^t m_{ao}} \tag{3}$$

We will call M_{ao} the transition matrix and m_{ao} the normalization vector. It is important to note that the parameters learned for this model will, in general, not be the same as those learned in the PSR model. We can no longer interpret the elements of x^t as probabilities; they may be negative or larger than 1. In TPSRs we seek to find a low dimensional state representation x^t that we can relate to the original PSR state r^t via a transformation matrix R, in particular $x^t = r^t R$. The transformation matrix allows us the added flexibility needed to tune the dimensionality of TPSRs.

3.1. Learning

Our learning algorithm begins with a set of action-observation histories B and a set of tests Q to examine. We make no assumption about the histories or tests except that they are numerous enough and different enough to exhibit the types of behaviors we wish to model. Write $p(q \mid bao)$ for the probability of success if we start from a fixed initial state, execute history b, take action a, receive observation o, and then run test q.

For each history $b \in B$, each action $a \in A$, each observation $o \in O$, and each test $q \in Q$, we estimate the probabilities $p(ao \mid b)$, $p(q \mid b)$, and $p(q \mid bao)$ by repeatedly setting the system up in a fixed initial state, executing the history b and running the tests ao, q, and aoq. This description implies that our system has a reset so that we can repeatedly put the system at the fixed initial state; in Section 4.2 we describe how we avoid this limitation in practice. After running the test we collect the probabilities $p(ao \mid b)$ into vectors C_{ao}, each with one element for every $b \in B$. We collect $p(q \mid b)$ into a matrix T_0 with one row for each $b \in B$ and one column for each $q \in Q$. And, we collect $p(q \mid bao)$ into a matrix T_{ao} with one row for each $b \in B$ and one column for each $q \in Q$. Finally, write T for the matrix obtained by stacking T_0 and T_{ao} for all a and o.

3.2. Learning the Transformation Matrix

Our learning algorithm for TPSRs proceeds in three steps: first we learn the transformation matrix R, then we learn the normalization vectors m_{ao}, and finally we learn the transition matrices M_{ao}.

Learning a transformation matrix R is equivalent to finding a low-dimensional representation of the belief states that result from executing each of the histories in B. We want a representation which has as few dimensions as possible, but which still allows us to predict the outcome of each test $q \in Q$ using linear weights m_q. We could use the rows of T to represent the belief states. This would trivially give us the ability to predict test outcomes (since the elements of T are already our predicted test outcomes). But, this is too rich a representation: Q may contain many irrelevant tests since we did not require the user to specify a minimal set of tests. So, in order to reduce the complexity of our representation to the inherent complexity of the problem, we employ singular value decomposition to find a low-dimensional representation that allows us to reconstruct T using linear weights: write

$$T = USV'$$

Now we can construct a transformation matrix R by selecting the first K columns of V:

$$R = V \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \begin{array}{l} \left.\vphantom{\begin{matrix}1\\0\\\vdots\\0\end{matrix}}\right\} K \\ \left.\vphantom{\begin{matrix}0\\\vdots\\0\end{matrix}}\right\} |Q| - K \end{array}$$

To learn an exact TPSR, we should pick K to be the number of non-zero singular values in S. In high-dimensional real-world systems, however, we may wish to pick a smaller K; doing so will produce a smaller TPSR at the cost of a possible loss of prediction quality (and in the worst case possible instability of long-term predictions). If we have too little training data, our most accurate predictions may come from reduced-size TPSRs, so we recommend cross-validation to choose the best size: we can learn a TPSR for each value of K, test its predictions on held-out data, and choose the size which produces the best predictions.

With this choice of R, our low-dimensional state representation $X = TR$ is equal to the first K columns of US. We can also write $X_0 = T_0R$ and $X_{ao} = T_{ao}R$.

3.3. Learning the Normalization Parameters

Now that we have a low-dimensional representation X, we can learn the parameters in our state-update equation (3). Recall that the denominator in (3), xm_{ao}, represents the probability of observing o if we perform action a in state x. We want to find the parameters m_{ao} which give the most accurate predictions of these probabilities. Our training data allows us to learn these parameters by a simple linear regression: with infinite training data, we would be able to predict the elements of C_{ao} perfectly from the elements of X_0 for each a and o. With finite training data there will be errors in learning X_0 and residual variance in C_{ao}, so we will pick m_{ao} to minimize squared error:

$$m_{ao} = \operatorname*{argmin}_{m} \|X_0 m - C_{ao}\|_2^2 \tag{4}$$

3.4. Learning the Transition Parameters

Given the m_{ao} parameters learned in the previous section, the state update function (3) is linear in the remaining M_{ao} parameters. These, too, can be learned via linear regression. Write $d_{ao} = Xm_{ao}$ for the vector of (now-constant) denominators in (3). Write D_{ao} for the matrix with d_{ao} on its diagonal. With this notation, we can write one copy of equation (3) for each $b \in B$:

$$X_{ao} = D_{ao}^{-1} X_0 M_{ao} \tag{5}$$

This equation will hold exactly only if we have infinite training data (so that our estimates of R and m_{ao} are perfect). With finite training data it will not generally be possible to select M_{ao} to achieve equality in (5), so we choose M_{ao} by minimizing squared error:

$$M_{ao} = \operatorname*{argmin}_{M} \|D_{ao}X_{ao} - X_0 M\|_F^2 \tag{6}$$

Here $\|\cdot\|_F^2$ is the sum of squared elements of a matrix. Note that we have premultiplied both sides of (5) by D_{ao} so that small errors in estimating D_{ao} don't result in large changes in the regression problem.

Note that each of the three steps of this algorithm produces a single well defined answer so our algorithm has no trouble with local minima. Also note that our algorithm does not take the structure of the TPSR parameters into account. It could learn parameters that do not produce valid probabilities. In practice, however, simply renormalizing produces excellent results. See Section 5 for further discussion of both of these details.

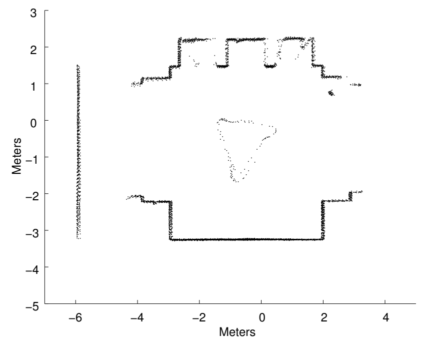


Figure 1. An overhead 2D map of the corridor where the robot was run. The triangular path of dots in the center shows the progress of the robot around one cycle.

4. Experimental Results

The main purpose of our experimental evaluation was to investigate the soundness of our learning technique and the performance of the TPSRs it produces. To this end, we performed a series of experiments on a robotic prediction task with a fixed policy and compared our learning algorithm for TPSRs to EM for learning POMDPs. (Because of the fixed policy, learning a POMDP in this case is equivalent to learning an HMM, and the EM algorithm is Baum-Welch.) As will be discussed below, we find that in many cases our TPSRs outperform HMMs learned by Baum-Welch for prediction. We also find that our learning algorithm produces good predictive models even when we restrict the dimensionality of its TPSRs to be much lower than the true dimensionality of the dynamical system. Unfortunately, Baum-Welch is not always the best algorithm for learning HMMs; we chose it because it is a standard approach with good freely available implementations. Because Baum-Welch is not the best known algorithm the scope in which our results can be interpreted is limited. In particular we cannot determine whether low dimensional HMMs are not as good for modeling our data as low dimensional TPSRs, or if we are just not learning the best HMM. In Section 6 we will outline a set of future experiments that would help separate these two concerns. Despite this important caveat, we believe that our results show that TPSRs perform well in comparison to standard methods and can be efficiently employed on real-world data.

4.1. Experimental Setup

We employed a Pioneer-class robot (see Figure 2) equipped with a laser range finder to collect data in an indoor environment. A 2D overhead map of the space is shown in Figure 1. We moved the robot repeatedly counter-clockwise in a triangular trajectory around the space (see the triangular path in the middle of the figure) and recorded its range

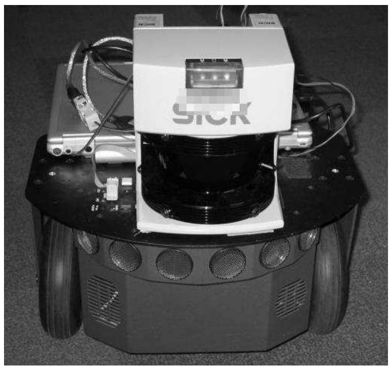


Figure 2. The pioneer robot used to collect the data.

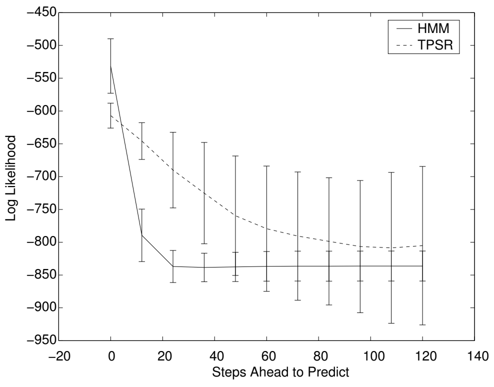


Figure 3. This graph shows the performance of HMMs trained by Baum-Welch and TPSRs as they are asked to predict further into the future. TPSRs give superior performance under most horizon lengths.

measurements. Each scan from the laser range finder consists of 180 range measurements along bearings spaced approximately 1 degree apart so that the scan covers 180 degrees in front of the robot. The robot collects scans several times a second and there were approximately 62 scans in the time it took to move around a single triangular cycle. In order to simplify our sensor model somewhat and produce a discrete set of possible observations we selected a subset of 124 scans at random from the collected data to serve as canonical scans. We then labeled each scan from the full data set with a number corresponding to which of the canonical scans it was closest to (in L_1 distance). This resulted in a discrete observation space where each observation is a number between 1 and 124.

4.2. Training and Testing Procedure

After preparing the data we split it into training and test sets and used the former to learn TPSR and HMM models. For HMMs, we randomly initialized the model parameters and ran Baum-Welch until it converged to a local minimum.

For TPSRs we needed to specify a set of action-observation histories B and a set of tests Q so that we could construct the training data matrices T_0, T_{ao}, and C_{ao}. For the tests Q, we included tests similar to the e-tests introduced by Rudary and Singh (2003). Our tests were all of the form "take N steps then test whether we observe Y," and to be sure we had an approximately-sufficient statistic we included all such tests for N up to 124 and all possible observations Y (a total of 15,376 tests in all). It is interesting that, although there are more than 15,000 tests in our set Q, we are able to achieve accurate predictions with 20-dimensional TPSRs and reasonable predictions with as few as six dimensions.

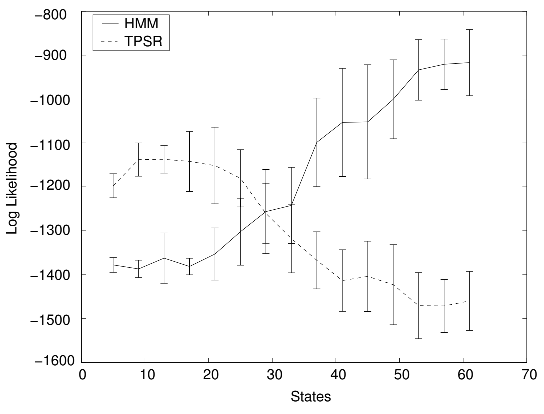
We included in B one history for each time step in our training data. This choice of B means that we have at most one chance to observe any given test or transition from each belief b, so our training data matrices contain only zeros and ones.[1] We chose this training procedure for two reasons: first, it avoids the necessity to reset the robot's state manually (which would have made the collection of training data slower). And second, we were curious whether our learning algorithm would still work with very noisy estimates of T_0, T_{ao}, and C_{ao}.

After training the models, we started each one in a random valid initial belief state and replayed observations from the test set, using the proper tracking equations for each model type to maintain belief states. We then asked each model to simulate forward and predict the distribution of observations some number of steps in the future. We explored the result of varying both the dimensionality (the number of states in the HMM and the number of linear combinations of tests in the TPSR) and the prediction horizon (how many steps ahead we asked the models to predict).

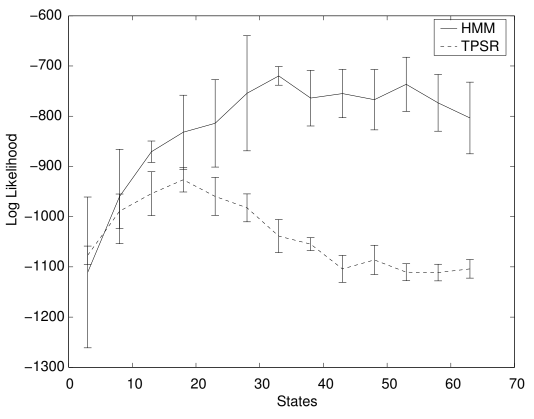
4.3. Results

In our first experiment we held the dimensionality of both models fixed at 20 and varied the predictive horizon from 0 (predict the next observation) to 60 (predict the observation we will receive 60 steps in the future). We performed 10-fold cross validation and measured the log likelihood of the true test set observations according to the model predictions. The results are shown in Figure 3; the error bars represent one standard deviation of the ten runs. For very short horizons the Baum-Welch trained HMM performed at least as well as the TPSR and for very long horizons both models tend toward uniform prediction. For much of the intermediate range the TPSR performs better.

[1]In fact, because we are executing a fixed policy, we observe each test exactly once. But, we only observe the transition from x_t to x_{t+1} for the observation o_t which actually occurred. The missing observations mean that, when learning M_{ao} in equation (6), we leave out time steps where o was not observed.



(a)



(b)

Figure 4. This graph shows the performance of HMMs and TPSRs as the dimensionality of each representation grows. (a) Shows the results for a fixed prediction horizon of31; TPSRs perform better than Baum-Welch trained HMMs for low dimensional representations. Panel (b) gives results for a fixed prediction horizon of 0, in this case the Baum-Welch trained HMMs perform at least as well as TPSRs regardless of dimension, and much better in most cases.

In a second experiment we held the predictive horizon fixed at 31 steps and varied the dimensionality of both models from 3 to 62. Again we performed cross validation and measured the log likelihood of the true observations under the models' predictions. The results (Figure 4(a)) show that TPSRs outperform Baum-Welch trained HMMs for very low dimensionality and reasonably long prediction horizons. From that figure it seems that high dimensional HMMs produce better predictions than the best TPSRs even with a prediction horizon of 31. However, as illustrated in the next section, we have found that the difference in likelihood between the best TPSRs and the best HMMs does not translate into a practical performance difference. Specifically, the TPSR with 20 dimensions gives qualitatively similar predictions to the HMM with 62 dimensions, but the HMM with 20 dimensions is qualitatively much worse than the TPSR with 20 dimensions. For completeness we have included Figure 4(b) which shows that for very short prediction horizons TPSRs are at a significant disadvantage when compared with HMMs.

4.4. Qualitative Comparison

Figure 5 shows several maps reconstructed from the predictions of our Baum-Welch trained HMMs and TPSRs. In order to generate the maps we first localized the robot manually so we could attach a true position to each time step in a range of test data. Then we used the output of our trained models to predict the most likely observation for each step and look up the canonical scans associated with those observations. We then plotted these scans as if they had been taken at the true positions of the robot. If the model predictions are good, then the plotted scans should line up to produce a map which is fairly clear and recognizable. Note, however, that this procedure is not capable of producing a perfect map: a perfect map would require that our set of canonical scans contained a scan from each true position of the robot, and since the canonical scans were selected at random from the training data and the robot moves noisily, this will not happen.

The first column in Figure 5 shows the performance of the Baum-Welch trained HMM and TPSR with 62 dimensions trying to predict the immediate observation on each time step. The second column shows the performance of the two models with 20 dimensions, but still predicting immediate observations; the third column shows the two 20-dimensional models trying to predict 31 steps in the future. These plots generally reinforce the results shown in Figures 3 and 4: the TPSR does particularly well with fewer dimensions and looking farther into the future. As mentioned above, though, one area of disagreement is apparent: Figure 4 showed a high dimensional HMM performing better than the best TPSR. From the maps of Figure 5 it is clear that the difference in prediction quality between the best TPSR and the best HMM we found is barely discernible. However, the difference between the low dimensional TPSR and the low dimensional HMM is substantial.

5. Related Work

PSRs were first introduced by Littman et al. (2001). In that paper they described PSRs and showed how to generate linear PSRs from POMDP models. They also demonstrated that nonlinear PSRs have the potential to be exponentially more compact than POMDPs. This work was followed up by Singh et al. (2003) who gave a gradient decent algorithm for learning the parameters of a linear PSR online. In con-

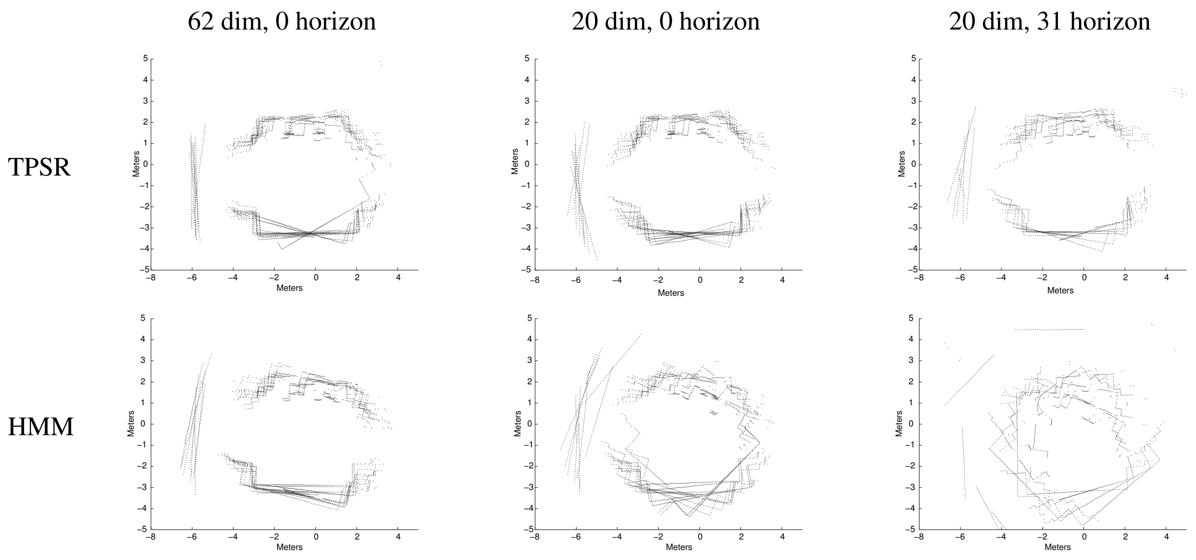


Figure 5. We have trained a variety of TPSR and HMM models and used their maximum likelihood predictions to plot maps. The top row of maps were produced by TPSRs and the bottom by HMMs. The first columns allows each model 62 dimensions and asks them to predict the next observation they will see. The second column is the same except only 20 dimensions are allowed. In the third column 20 dimensions are allowed and the models are asked to predict 31 steps into the future.

trast to that work, our approach provides a batch algorithm for learning PSR parameters, and provides a mechanism for tuning the dimensionality of the representation given a large set of tests. Previous dimensionality-selection results have used techniques such as forward selection to identify such a set.

One of the main reasons that PSRs have attracted interest, and one of the motivations behind this work, is their potential for being more compact than POMDPs. Though there have been examples of linear PSRs that are more compact than POMDPs in the literature(Jaeger, 2000), there are no published learning procedures for finding them from data. Rudary and Singh (2003) developed an algorithm for learning a certain class of non-linear PSRs, and in the noise free case learned a non-linear PSR that was more compact than the minimal POMDP model. Our work is limited to linear PSRs, but we provide a way to tune the dimensionality of the representation, even to values too low to represent the dynamical system exactly. We have shown that, even when our representation is far too small, we can predict well.

One of the motivations for finding compact representations is the hope that they will result in easier planning problems. Planning in POMDPs has proven difficult because planning is usually exponential in the representation size. In the literature to date only a single planning algorithm has been published for PSRs (Izadi & Precup, 2003) so there is a great need for exploration in this area.

PSRs are related to other models of dynamical systems such as HMMs. There are several important differences between TPSRs and HMMs. One important difference, noted previously by Littman et al. (2001), is that PSRs learn based only on observable quantities, namely the outcomes of tests. HMMs on the other hand must learn based on hidden quantities: for example, they must estimate state transitions even though they cannot observe the states.

Learning a POMDP or HMM from observations of a physical process is difficult because of the problem of local minima. For example, one popular algorithm for learning these models is expectation-maximization or EM (known as Baum-Welch in the case of HMMs). In EM, we alternate between two steps: first we fix the parameters of our model and estimate the expected sequence of states that explains our training data, and then we fix the expected state sequence and optimize our model parameters. It is easily possible for EM to get stuck at a local minimum where the state sequence and model parameters are consistent with each other but not optimal.

In sharp contrast, our TPSR learning procedure has no local minima: each of its three steps has a single, well-defined answer (after normalizing the representation of the SVD for sign changes and order of the singular values). And (after normalizing), the output of each step is continuous in its input[2]; so, as the amount of training data increases, the true dimensionality of the TPSR becomes apparent and the learned parameters smoothly approach their optimal values.

[2]This is not precisely true if there are repeated singular values in the SVD, but a similar statement still holds in that case.

The disadvantage of our learning procedure is that it pays no attention to the structure of TPSR parameters. Not every set of parameters M_{ao} and m_{ao} corresponds to a valid TPSR; for example, poorly-chosen parameters can cause the update equation (3) to lose the normalization of our state vector x so that predicted observation probabilities xm_{ao} become negative or don't sum to 1. In practice this disadvantage means that, because of small errors in estimating T and C_{ao}, the TPSR learned by our algorithm will only produce approximately-normalized probability predictions. Interestingly, even when we observe unnormalized or negative probability predictions in our experiments, simply renormalizing produces excellent results.

Another important difference between HMMs and PSRs is that HMMs rely strongly on the Markov assumption. That is, they assume that the current state of the HMM completely specifies the state of the underlying system. This assumption makes HMMs very sensitive to modeling error: if the state doesn't fully capture the dynamical system then they can quickly become overconfident and make bad predictions. In contrast, PSRs directly attempt to predict the outcome of long-term tests and so seem to be less reliant on short-term independence assumptions.

6. Conclusion and Future Work

We have developed a new PCA-based algorithm for learning TPSR parameters from data. Our algorithm also partially addresses the problem of discovery by allowing users to provide only a large sufficient set of tests which is then reduced to a smaller set via SVD. We have shown that our algorithm can successfully learn TPSRs from data in real world domains. Further we have shown that under a range of circumstances TPSRs can outperform HMMs trained by Baum-Welch, especially in cases where model dimensionality is limited or the prediction horizon is long.

An important line of future research will involve determining whether PSRs, and specifically TPSRs, work better for prediction in long horizon and restricted dimensionality situations than HMMs in general, or if our Baum-Welch training method was simply inadequate to find a good HMM model. In order to separate these two concerns a series of experiments should be performed that compare TPSRs against more modern methods such as the U-Tree algorithm (McCallum, 2000) or Bayesian model merging (Stolcke & Omohundro, 1994). An even more convincing experiment would pit our learning algorithm against an optimal low-dimensional HMM if a simple example could be found where the optimal HMM was easy to produce.

Another important extension to our experimental work would be to measure performance in predicting off-policy test outcomes. With off-policy predictions, we could explore planning algorithms to try to take advantage of the good predictive ability we have seen in low dimensional TPSRs. Another important direction is to alter our training procedure so that it can guarantee that its learned TPSRs will always produce normalized probability predictions.

There are many future directions left to be explored, but the present paper shows, possibly for the first time, that predictive representations are promising in real-world robotics tasks. We find this result remarkable given that recent work in robotics has predominantly relied on generative HMM-style models. Despite the considerable remaining tasks, we hope that this work establishes a new and effective learning algorithm that makes predictive representations applicable to real-world robotics problems.

References

Cassandra, A., Littman, M. L., & Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes. *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI).*

Cassandra, A. R., Kaelbling, L. P., & Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI).*

Izadi, M. T., & Precup, D. (2003). A planning algorithm for predictive state representations. *The Eighteenth International Joint Conference on AI (IJCAI).*

Jaeger, H. (2000). Observable operator models for discrete stochastic time series. *Neural Computation*, *12*, 1371–1398.

Littman, M., Sutton, R., & Singh, S. (2001). Predictive representations of state. *Advances In Neural Information Processing Systems (NIPS).*

McCallum, A. K. (2000). *Reinforcement learning with selective perception and hidden state*. Ph.d. thesis, Department of Computer Science, University of Rochester.

Rudary, M., & Singh, S. (2003). A nonlinear predictive state representation. *Advances In Neural Information Processing Systems (NIPS).*

Singh, S., Littman, M. L., Jong, N. E., Pardoe, D., & Stone, P. (2003). Learning predictive state representations. *The Twentieth International Conference on Machine Learning (ICML).*

Stolcke, A., & Omohundro, S. M. (1994). *Best-first model merging for hidden Markov model induction* (Technical Report TR-94-003). 1947 Center Street, Berkeley, CA.