Draft: Please do not distribute

Learning Predictive State Representations

Satinder Singh
Department of Electrical Engineering and Computer Science
University of Michigan
baveja@eecs.umich.edu

Michael L. Littman
Department of Computer Science
Rutgers University
mlittman@cs.rutgers.edu

Richard Sutton
Stow Research
Chester, New Jersey
rich@richsutton.com

Peter Stone
Department of Computer Sciences
The University of Texas at Austin
pstone@cs.utexas.edu

Abstract

We introduce the first algorithm for learning predictive state representations. Predictive state representations, or PSRs are a way of representing the state of a controlled dynamical system in terms of predictions of *tests*, where tests are sequences of actions and observations said to be true if and only if all the observations occur given that all the actions are taken. The problem of finding a good PSR—one that is a sufficient statistic for the dynamical system—can be divided into two parts: 1) *discovery* of a good set of tests, and 2) *learning* to make accurate predictions for those tests. In this paper, we address the learning part of the problem for linear PSRs and show that our algorithm makes correct predictions in several sample systems.

1 Introduction

Predictive state representations (PSRs; Littman, Sutton, & Singh, 2002) are a new way, based on previous work (Jaeger, 2000, Rivest & Schapire, 1994), for representing the state of a controlled dynamical system from its history of observations and actions taken. A distinctive feature of PSRs is that the representation is not interpreted as a memory of past observations, or as a distribution over hidden states, but as a vector of prediction probabilities. For example, the first component of the vector might be the probability that observation *1* will occur if action a is taken, and the second component of the vector might be the probability that observation *1* will occur twice in succession if action a is taken followed by action b, and so on. For each component of the vector, there is a sequence of alternating actions and observations called a *test*. These tests, called the *core tests*, define the PSR.

In previous work, we have shown that PSR representations are more general than methods based on a fixed-length history or window of past experience (k-Markov models, state is most recent k action–observation pairs), and are as general and at least as compact as partially observable Markov decision process (POMDP) representations.

Another reason for interest in PSRs is that their structures—the predictions—are directly related to observable quantities and thus may be easier to learn than, for example, POMDP representations. Analogous demonstrations have been made for the diversity representation (Rivest & Schapire, 1994) and the observable operator representation (Jaeger, 2000). Both of these representations are similar to but more restrictive than PSRs. In this paper, we use the term *learning* to refer to the process of finding how to maintain correct predictions for a given set of tests. The other, perhaps larger, problem of choosing which tests to use to define the representation, which might be called the *discovery* problem, we postpone to another time.

2 Predictive State Representations

The dynamical system that we are trying to model and predict is assumed to operate at discrete time intervals, to receive actions from a finite set A, and to produce observations from a finite set O. The source of the actions, the *behavior policy*, is at this point arbitrary. The system and the behavior policy together determine a probability distribution over all histories (sequence of alternating actions and observations from the beginning of time).

PSRs are based on the notion of tests. A test q is a finite sequence of alternating actions and observations, $q \in \{\mathcal{A} \times \mathcal{O}\}^*$. For a test $q = a^1 o^1 a^2 o^2 \ldots a^l o^l$, its *prediction* given some history $h = a_1 o_1 a_2 o_2 \ldots a_t o_t$, denoted $p(q|h)$, is the conditional probability of seeing the test's observations in sequence given that the test's actions are taken in sequence from history h:

$$p(q|h) = prob(o_{t+1} = o^1, o_{t+2} = o^2, \ldots o_{t+l} = o^l | h, a_{t+1} = a^1 \ldots a_{t+l} = a^l).$$

For completeness, we define the prediction for the null test to be one. Given a set of n tests $Q = \{q_1, \ldots, q_n\}$, its *prediction vector*, $p(Q|h) = [p(q_1|h), p(q_2|h), \ldots, p(q_n|h)]$, has a component for the prediction for each of its tests. The set Q is a predictive state representation (PSR) if and only if its prediction vector forms a sufficient statistic for the dynamical system, that is, if and only if

$$p(q|h) = f_q(p(Q|h)), \tag{1}$$

for any test q and history h, and for some *projection functions* $f_q : [0,1]^n \mapsto [0,1]$. This means that the predictions of the tests in Q can be used to calculate a prediction for any other test. The tests in set Q are called *core tests* as they constitute the foundation of the representation of the system. In this paper, we focus on *linear* PSRs, for which the projection functions are linear—there exists a *projection vector* m_q, for every test q, such that

$$p(q|h) = f_q(p(Q|h)) = p(Q|h)^T m_q, \tag{2}$$

for all histories h. Let q_i denote the ith core test in a PSR. Its prediction can be updated recursively from the prediction vector, given a new action–observation pair a, o, by

$$p(q_i|hao) = \frac{p(aoq_i|h)}{p(ao|h)} = \frac{p(Q|h)^T m_{aoq_i}}{p(Q|h)^T m_{ao}}. \tag{3}$$

As an example of PSRs, consider the dynamical system depicted in Figure 1. It consists of two actions and two observations and can be described by a linear string of 5 "underlying" states with a distinguished *reset* state on the far right. Action *f* causes the system to move uniformly at random to the right or left by one state, bounded at the two ends. Action *r*

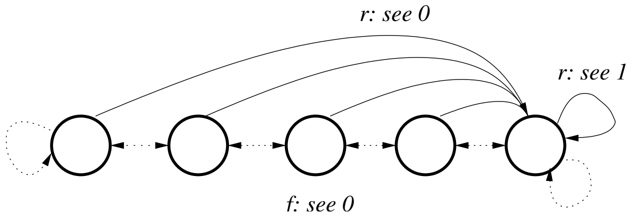

Figure 1: An example dynamical system from Littman et al. (2002).

causes a jump to the reset state irrespective of the current state. The observation is always *0* unless the *r* action is taken when the system is already in the reset state, in which case the observation is *1*. As shown by Littman et al. (2002), for this system, the prediction vector for the 5 core tests *f 0*, *r 0*, *f 0 r 0*, *f 0 f 0 r 0*, and *f 0 f 0 f 0 r 0* constitutes a linear PSR. Completing this example, consider how the components of the prediction vector might be updated on action–observation *f 0*. Most of these updates are quite simple because the tests are so similar: the new prediction for *r 0* is exactly the old prediction for *f 0 r 0*, for example. The only non-trivial test is *f 0 f 0 f 0 r 0*. We derived that its prediction can be computed from $0.250\ p(r\,0|h) - 0.0625\ p(f\,0\ r\,0|h) + 0.750\ p(f\,0\,f\,0\ r\,0|h)$. This example illustrates that the projection vectors need not contain only positive entries, which makes for a challenging learning problem.

In deriving a learning algorithm, a crucial fact to notice from Equation 3 is that to update the predictions of the core tests, it is only necessary to have predictions for all *1-step extensions* of all the core tests (the numerator on the right-hand side of Equation 3) and the 1-step extensions of the null test (the denominator on the right-hand side of Equation 3). Collectively, we call these 1-step extensions of the core and null tests *extension tests* and denote the set of them X. There is a separate projection vector m_x for each extension test $x \in X$; collectively these projection vectors are the parameters of the PSR, and the learning problem we studied is to determine these parameters from data.

3 Learning Algorithm

To learn a linear PSR model of a dynamical system from experience, a learner must select the set of core tests Q, and thereby the set of extension tests X, and then set the vectors m_x for $x \in X$ to maintain correct predictions. In this section, we present an algorithm for the latter problem of estimating the PSR parameters. At each time step t, an action–observation pair a_t, o_t is generated from the dynamical system, advancing the history of data available by one step. The history prior to step t is denoted h_{t-1}. We assume the actions are chosen according to some known behavior policy π that is ϵ-soft, specifically $prob(a|h, \pi) \geq \epsilon > 0$ for all histories h and for all $a \in A$, to encourage exploration. A learning algorithm has to use the history so far to estimate the PSR parameters, as well as use these estimated parameters to maintain an estimated state representation. We will denote estimated parameter vectors by $\hat{m}$ and estimated prediction vectors by $\hat{p}$.

A test q is considered to be *executed* at time t if the sequence of actions starting at time t matches the sequence of actions in q, and the *outcome* of such a test, $\chi_{q,t}$, is 1 if the sequence of observations starting at time t match the sequence of observations in q, and is 0 otherwise. Our learning algorithm capitalizes on the fact that the history data provides outcomes for the extension tests that are executed. In practice, the algorithm will do an update for time-step t when it has a history of length $t + k$ where k is the length of the longest extension test. This way, the algorithm can look forward in time to determine the extension tests executed from t and their outcomes, and use them as training data for the PSR parameters.

Our learning algorithm updates the parameter vectors for all executed extension tests to make them more likely to predict their outcomes. The parameter vectors of extension tests that are not executed are not updated. Formally, letting E_t denote the set of extension tests that are executed at time t (note that all $x \in E_t$ must be of the form $a_t oq$ for some $q \in Q \cup \phi$, where ϕ is the null test), the update rule is:

$$\begin{aligned} \forall x \in E_t,\ \ \hat{m}_x &\leftarrow \hat{m}_x - \alpha_t \frac{1}{w^{\pi}_{x,t}}[\chi_{x,t} - \hat{p}_t^T \hat{m}_x]\hat{p}_t \\ \forall x \notin E_t,\ \ \hat{m}_x &\leftarrow \hat{m}_x, \end{aligned} \tag{4}$$

where $\hat{p}_t = \hat{p}(Q|h_{t-1})$ is the estimated state vector at time t, α_t is the step-size parameter (which can change over time), and $w^{\pi}_{x,t}$ is the importance sampling weight for extension test $x = a_t oq$ at time t. The importance sampling weight is defined as follows: assume without loss of generality that $q = a^1 o^1 a^2 o^2 \dots a^l o^l$, then

$$w^{\pi}_{x,t} = \Pi^{l}_{i=1} prob(a^i | h_{t+i}, \pi),$$

where h_{t+i} is the history at time $t+i$. For the special case of $x = a_t o$, the importance sampling weight $w^{\pi}_{x,t} = 1$. These weights capture the likelihood of selecting various sequences of actions and are needed to compensate for unbalanced exploration. The updated parameters are then used to compute the next estimated state vector $\hat{p}_{t+1} = \hat{p}(Q|h_t)$ using Equation 3 (where the true parameters are replaced by the estimated parameters). We upper and lower bound the entries of the $\hat{p}_{t+1}$ vector by clipping them at 1 and some $\eta > 0$ if necessary (not doing so can make the learning rule unstable).

An interesting feature of our learning algorithm is that it uses errors in the prediction of outcomes of *multi-step* actions or tests. By contrast, most algorithms for learning models of POMDPs and k-Markov models use errors in the prediction of immediate observations. We believe that learning from multi-step predictions may be more robust and efficient.

At the same time, other multi-step prediction error algorithms are possible; in particular our learning algorithm ignores the effect of changing the parameters on the input $\hat{p}_t$ itself, and hence can be viewed as a "myopic" version of a more complex learning algorithm that would "unroll" the parameter changes backwards in time; in the appendix we show this relationship formally. Nevertheless, as in some other instances of myopic algorithms provably converging to desired quantities, what we would like to prove is that our learning algorithm converges asymptotically with probability one to parameters that maintain correct predictions if the set of core tests Q constitutes a PSR, the behavior policy is ϵ-soft, and the step-size sequence $\{\alpha_t\}$ converges to zero neither too quickly nor too slowly. While we cannot prove or disprove this conjecture yet, the next section provides empirical evidence of convergence on a few small problems.

4 Empirical Results

Our first results are on the simple dynamical system of Figure 1. We simulated float/reset, choosing actions uniformly randomly at each time step, and used the sequence of action–observation pairs to learn the parameters via Equation 3. Core tests are the ones given in Section 2. The error measured at time-step t is the squared error between the true one-step observation probabilities and the estimated observation probabilities. So, if the probability that the dynamical system would choose observeration o at time t given the history is $p(o,t)$ and the learned model estimates this probability as $\hat{p}(o,t)$, then the performance is $\sum_{t=1}^{T} \sum_o (p(o,t) - \hat{p}(o,t))^2 / T$. This measure has the useful property that a model has zero error if and only if it produces correct predictions. Even though the true one-step

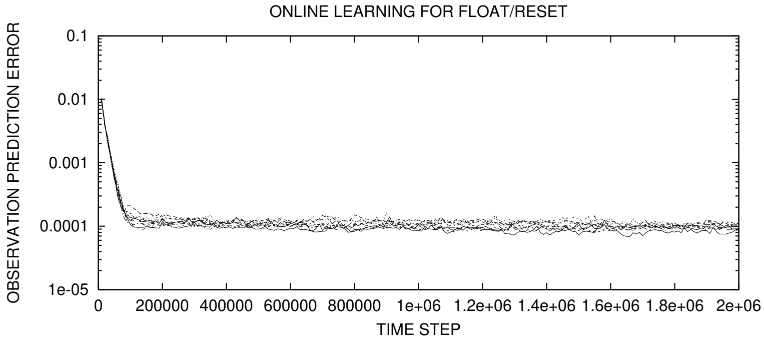

Figure 2: Prediction error for ten runs of our learning algorithm on float/reset.

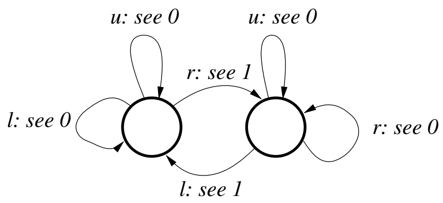

Figure 3: A simple two-state test system that no k-Markov model can represent.

observation probabilities are not generally available, in this case we can use our knowledge of the underlying system parameters to compute them for the purposes of measuring the error.

We computed the average error per time step over intervals of $100,000$ steps and plotted the sequence of errors for 10 runs in Figure 2 as learning curves. For each run, the step-size sequence started at 0.1 and was halved every $100,000$ steps to a lower-bound of 0.00001. The components of the estimated state vector were upper-bounded at 1.0 and lower-bounded at 0.0001. As can be seen, the error decreases to 0.0001 in each run. The error does not go to zero because the lower-bound clipping of the estimated state vectors prevents it from doing so. (In future work, we will explore shrinking this lower-bound to zero over time to allow for true convergence.)

Our second example is based on an extremely simple 3-action, 2-observation system, flip, illustrated in Figure 3. It can be defined using two states. The observation *1* is made when the state changes, and *0* is made otherwise. Action *u* keeps the state the same, action *l* causes a deterministic transition to the left state, and action *r* causes a deterministic transition to the right state. The example was inspired by one used by Finney et al. (2002) that proved nettlesome to a history-window-based approach.

Because the *u* action provides no state information, the higher the probability of choosing it, the less effective a k-Markov model will be in capturing the state information. At the same time, the higher the probability of choosing *u*, the less important it is to track the state, as most of the time the true observation will be *0*. Analytically, if U is the probability that *u* is chosen on any step, a 1-Markov model will exhibit an error of $U/2 - U^2/2$ (the model will have to guess randomly, and be wrong 50% of the time, whenever *u* action is followed by a either *l* or *r*). Error is maximized for a 1-Markov model when $U = 1/2$.

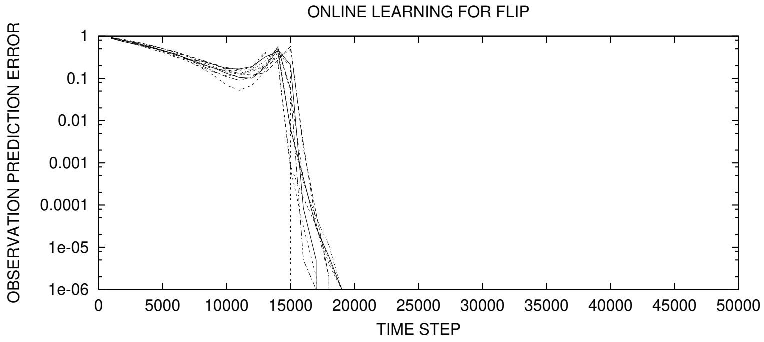

Figure 4: Prediction error for ten runs of our learning algorithm on flip.

Error decreases for k-Markov models with increasing k, but it can never be driven to zero because there is always some probability of a sequence of *u* actions filling up the history window.

We computed the average error per time step over intervals of 1000 steps and plotted the sequence of errors for 10 runs in Figure 4 as learning curves. The behavior policy chose *u* with probability 0.5. Once again, our learning algorithm is able to reduce error over a series of learning trials.

The experiments described above demonstrate empirically that our learning algorithm converges to maintaining perfect predictions on two example problems, when run online. Next, we provide some evidence that it can also behave well in a more classical "batch" learning setting.

We simulated flip, choosing actions according to the same behavior policy described above (*u* with probability 1/2, and each of *l* and *r* with probability 1/4). We used sequences of 1000 action–observation pairs as a batch training set and another 100,000 to serve as a testing set[1] As before, error is measured by the sum of squared differences between the 1-step observation predictions from the learned model and those of the true model.

We used a constant step-size parameter of 0.1 and experimented with two different sets of core tests: two tests that constitute a sufficient statistic (chosen using prior knowledge of the system; Littman et al., 2002) and the set of all 1-step tests (also a sufficient statistic) and the results were qualitatively identical. In 100 runs, the algorithm's testing error never exceeded 0.0001 and the median error was 4.0×10^{-8}; our learning algorithm learned a correct model of the dynamical system every time.

In contrast, we repeatedly simulated k-Markov models on this example and obtained average errors of 0.125 (1-Markov), 0.0625 (2-Markov), and 0.0312 (3-Markov). It is interesting to note that the parameters of a 1-Markov model and a PSR with all 1-step tests as core tests are very similar; both consist of prediction information for each action–observation pair. However, a 1-Markov model looks backward and has per-step error of 0.125 on flip, whereas the corresponding PSR looks forward and represents flip perfectly.

We also tried to learn POMDP representations for flip using a direct implementation of the EM algorithm[2], first proposed for POMDPs by Chrisman (1992). We attempted to learn

[1] As in the PAC setting, we used the same behavior policy for training and testing.

[2] Our experiments used Kevin Murphy's Bayes Net Toolkit: `http://www.cs.berkeley.edu/~murphyk/Bayes/bnt.html`. EM ran for a maximum of 100 iterations or until two suc-

models with 4, 5, 6, 7, or 8 states. The learning code we used learned POMDPs in which observations are only conditioned on states, and so a 4-state model is needed to represent flip (one for the left state arriving from the right, one for the left state otherwise, one for the right state arriving from the left, and one for the right state otherwise). In 25 learning runs, EM learned a correct 4-state model 48% of the time, 5-state model 64% of the time, 6-state model 84% of the time, 7-state model 95% of the time, and 8-state model 100% of the time. For consistent results that avoid local optima, EM requires far more than the minimum number of required states. Because the hidden states of a POMDP can only be trained indirectly, EM quite often fails to find appropriate models.

Note that there are more sophisticated algorithms for learning POMDP models (Nikovski, 2002) that need to be compared systematically to our learning algorithm; our purpose in presenting these preliminary results is to demonstrate that our learning algorithm is indeed solving a non-trivial task.

5 Conclusion

PSRs of dynamical systems are as general as POMDP representations, but have the advantage that they are grounded in actions and observations. Littman et al. (2002) suggested that this advantage may lead to more efficient learning and greater robustness to local optima. In this paper, we presented the first learning algorithm for PSRs and showed empirically on small problems that it converges to a model that maintains correct predictions.

We have just begun our investigations of learning and discovery of predictive state representations in dynamical systems. We intend to expand the range of systems for which we have computational and comparative experience, develop an asymptotic analysis of our learning algorithm, integrate PSRs with reinforcement learning and planning, and expand the notion of PSRs to a broader range of knowledge.

Appendix

We now derive a multi-step prediction error gradient algorithm for learning the PSR parameters from history data, and show that our learning algorithm is a myopic version of the gradient algorithm. Consider the following multi-step prediction error function:

$$\begin{aligned}\text{Error(t)} &= \sum_{x \in X_{|a_t}} [p(x|h_{t-1}) - \hat{p}(x|h_{t-1})]^2 \\ &= \sum_{x \in X_{|a_t}} [p(x|h_{t-1}) - \hat{p}^T(Q|h_{t-1})\hat{m}_x]^2, \end{aligned} \tag{5}$$

where $X_{|a_t}$ are all the extension tests that begin with action a_t. The gradient algorithm will iteratively move the parameters in the direction of the gradient. The gradient of the error function with respect to the parameter vector m_x is as follows: for $x \in X_{|a_t}$

$$\begin{aligned}\frac{\partial \text{Error}(t)}{\partial m_x} &= -\Big(2[p(x|h_{t-1}) - \hat{p}^T(Q|h_{t-1})\hat{m}_x] \\ &\qquad [\hat{p}(Q|h_{t-1}) + \frac{\partial \hat{p}(Q|h_{t-1})}{\partial m_x}\hat{m}_x]\Big) \end{aligned} \tag{6}$$

while for $x \notin X_{|a_t}$

$$\frac{\partial \text{Error}(t)}{\partial m_x} = -\Big(2[p(x|h_{t-1}) - \hat{p}^T(Q|h_{t-1})\hat{m}_x]\frac{\partial \hat{p}(Q|h_{t-1})}{\partial m_x}\hat{m}_x\Big). \tag{7}$$

cessive iterations changed the log likelihood by less than 0.0001.

Unfortunately, the gradient of the error function is complex to compute because the derivative of the estimated state, $\frac{\partial}{\partial m_x}\hat{p}(Q|h_{t-1})$, requires taking into account the entire history (or much of it, depending on the mixing time). We will instead be myopic and take the estimated state as "given", that is, assume the derivative of the estimated state with respect to any of the parameters to be zero. In that case, we can define the following correction term: for all $x \in X$

$$D_x(t) \quad = \quad -\chi_{x \in X_{|a_t}} \Big(2[p(x|h_{t-1}) - \hat{p}^T(Q|h_{t-1})\hat{m}_x]\hat{p}(Q|h_{t-1}) \Big), \tag{8}$$

where $\chi_{x \in X_{|a_t}}$ is an indicator function that is 1 if $x \in X_{|a_t}$ and is 0 otherwise. Of course, we cannot use $D_x(t)$ because we do not have access to $p(x|h_{t-1})$. Instead, we have access to $\chi_{x,t}$ with probability $w_{x,t}$, and so we instead use the stochastic correction term

$$d_x(t) \quad = \quad -\chi_{x \in X_{|a_t}} \frac{1}{w_{x,t}} \Big(2[\chi_{x,t} - \hat{p}^T(Q|h_{t-1})\hat{m}_x]\hat{p}(Q|h_{t-1}) \Big) \tag{9}$$

in our learning algorithm. It can be seen that $E_{|\pi,h_{t-1},a_t}\{d_x(t)\} = D_x(t)$, where $E_{|\pi,h_{t-1},a_t}$ is expectation given behavior policy π, history h_{t-1}, and the action at time t. Therefore, the effect of the learning algorithm defined in Equation 4 is to follow the myopic approximation of the multi-step prediction error gradient, in expectation.

Acknowledgments: We thank Natalia Hernandez Gardiol and Daniel Nikovski for pointers to relevant materials.

References

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 183–188). San Jose, California: AAAI Press.

Finney, S., Gardiol, N. H., Kaelbling, L. P., & Oates, T. (2002). *Learning with deictic representation* (Technical Report AIM-2002-006). MIT AI Lab.

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

Littman, M. L., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. *Advances in Neural Information Processing Systems 14*. In press.

Nikovski, D. (2002). *State-aggregation algorithms for learning probabilistic models for robot control*. Doctoral dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Rivest, R. L., & Schapire, R. E. (1994). Diversity-based inference of finite automata. *Journal of the ACM*, *41*, 555–589.