Learning Predictive State Representations

Satinder Singh BAVEJA@EECS.UMICH.EDU
Computer Science and Engineering, University of Michigan

Michael L. Littman MLITTMAN@CS.RUTGERS.EDU
Department of Computer Science, Rutgers University

Nicholas K. Jong NKJ@CS.UTEXAS.EDU
David Pardoe DPARDOE@CS.UTEXAS.EDU
Peter Stone PSTONE@CS.UTEXAS.EDU
Department of Computer Sciences, The University of Texas at Austin

Abstract

We introduce the first algorithm for learning predictive state representations (PSRs), which are a way of representing the state of a controlled dynamical system. The state representation in a PSR is a vector 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 present detailed empirical results using a gradient-based algorithm for addressing the second problem. Our results demonstrate several sample systems in which the algorithm learns to make correct predictions and several situations in which the algorithm is less successful. Our analysis reveals challenges that will need to be addressed in future PSR learning algorithms.

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 a history of actions taken and resulting observations. 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 window of past experience (like k-Markov models, in which the state is the 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 hidden-state representations like POMDPs. Analogous demonstrations have been made for the diversity representation (Rivest & Schapire, 1994) and the observable operator representation (Jaeger, 2000). 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.

In previous work (Littman et al., 2002), we introduced

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

PSRs as a new representation suitable for learning. This paper introduces the first concrete learning algorithm for PSRs in POMDPs. We present detailed empirical results showing that our gradient-based algorithm makes correct predictions in several sample systems. We also present and analyze some systems and situations in which the algorithm is less successful and analyze the reasons for its failures. Overall, our results suggest that our learning algorithm is a good starting point for learning PSRs. The primary goals of this paper are to introduce the initial PSR learning algorithm, to analyze its empirical performance in detail, and to suggest directions for future development of PSR learning algorithms.

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 (sequences 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 $q = a^1 o^1 a^2 o^2 \dots a^l o^l$, its *prediction* given some history $h = a_1 o_1 a_2 o_2 \dots a_t o_t$, 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: $p(q|h) = prob(o_{t+1} = o^1, \dots, o_{t+l} = o^l | h, a_{t+1} = a^1 \dots a_{t+l} = a^l)$. For completeness, we define the prediction for the null test to be one. Given a set of n tests $Q = \{q_1, \dots, q_n\}$, its *prediction vector*, $p(Q|h) = [p(q_1|h), p(q_2|h), \dots, p(q_n|h)]$, 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

$$p(q|h) = f_q(p(Q|h)), \qquad (1)$$

for any test q and history h, and for some *projection functions* $f_q : [0,1]^n \mapsto [0,1]$. This means that for all histories 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

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

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

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

where we used the fact that aoq_i and ao are also tests.

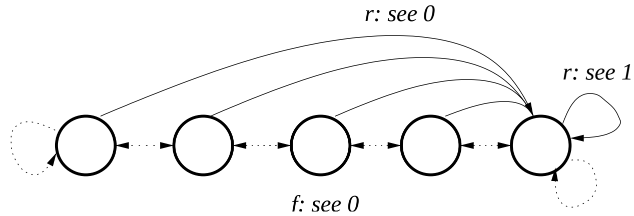

Figure 1. The float/reset example problem.

To illustrate these concepts, consider the dynamical system in Figure 1, consisting of two actions and observations and described by a linear chain of 5 environmental states with a distinguished *reset* state on the far right. Action f causes uniform motion to the right or left by one state, bounded at the two ends. Action r causes a jump to the reset state. The observation is 1 when the r action is taken from the reset state, and is 0 otherwise. Littman et al. (2002) showed the prediction vector for the 5 core tests $f0$, $r0$, $f0r0$, $f0f0r0$, and $f0f0f0r0$ constitutes a linear PSR for this system. Consider how the components of the prediction vector are updated on action–observation $f0$. Most of these updates are quite simple because the tests are so similar. For example, the new prediction for $r0$ is $p(f0r0|h)/p(f0|h)$, where both numerator and denominator are already part of the PSR. The nontrivial case is the update for the longest core test (see Littman et al. (2002) for details), $p(f0f0f0f0r0|h) = 0.250\ p(f0|h) - 0.0625\ p(r0|h) + 0.750\ p(f0f0r0|h)$, matching the form of Equation 3. This example shows that the projection vectors may contain negative entries, which makes for a challenging learning problem.

To derive a learning algorithm, notice from Equation 3 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 the equation) and the 1-step extensions of the null test (the denominator on the right-hand side of the equation). Collectively, we call these 1-step extensions of the core and null tests *extension tests* and denote the set of them X. For linear PSRs there is a separate projection vector m_x for each extension test $x \in X$. The m_x vectors constitute the parameters of the PSR, and the learning problem is to determine from data the parameters for a given set of core tests.

3. Learning Algorithm

In this section, we present our algorithm for learning the PSR parameters from data. 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 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 matches the sequence of observations in q, and is 0 otherwise. A learning algorithm has to use the history to estimate the PSR parameters, as well as use these estimated parameters to maintain an estimated predictive state representation. We will denote estimated parameter vectors by $\hat{m}$ and estimated prediction vectors by $\hat{p}$.

One way to derive a learning algorithm is to define an appropriate error function and then use the gradient of the error function with respect to the parameters to derive the learning rule. What is an appropriate error function for evaluating a model of a dynamical system? Whereas learning algorithms for history-window and belief-state approaches minimize prediction error for immediate or one-step observations, we instead use a *multi-step* prediction error function

$$\text{Error(t)} = \sum_{x \in X_t} [p(x|h_{t-1}) - \hat{p}(x|h_{t-1})]^2, \quad (4)$$

where X_t contains all extension tests possible from time t (i.e. those that begin with action a_t).

In the appendix, we argue that it is computationally difficult to directly use the true gradient of the error function and so instead we will use a simple and myopic approximation to the gradient in our learning algorithm. 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 $a_t o q$ for some $q \in Q \cup \phi$, where ϕ is the null test), the learning rule is:

$$\begin{aligned} \forall x \in E_t,\ \hat{m}_x &\leftarrow \hat{m}_x + \alpha_t \frac{1}{w^{\pi}_{x,t}} [\chi_{x,t} - \hat{p}_t^T \hat{m}_x] \hat{p}_t \\ \forall x \notin E_t,\ \hat{m}_x &\leftarrow \hat{m}_x, \end{aligned} \quad (5)$$

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[1] for extension test $x = a_t o q$ at time t. 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. The updated parameters are then used to compute the next estimated state vector $\hat{p}_{t+1} = \hat{p}(Q|h_t)$ using estimated parameters in Equation 3. We bound the entries of the $\hat{p}_{t+1}$ vector by clipping them at 1 and $\eta > 0$; we found that not doing so sometimes made the learning rule less stable.

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 to use as training data.

Of course, other multi-step prediction error gradient algorithms are possible; our learning algorithm ignores the effect of changing the parameters on the input $\hat{p}_t$ itself, and other algorithms that are less myopic and "unroll" the parameter changes backwards in time can be derived (see appendix). The performance of gradient-based learning algorithms depends, of course, on various properties of the error landscape: the presence or absence of local minima, the size of the basins of attraction and stability of the various minima, the noisiness of the stochastic gradient estimate, and so on. Our learning algorithm is the computationally simplest gradient-based rule to explore, and our empirical results help uncover properties of the error landscape.

4. Empirical Results

In this section, we present detailed empirical results designed to expose the strengths and weaknesses of our learning algorithm.

[1] The importance sampling weight compensates for unbalanced exploration within tests and 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 are only relevant if the behavior policy is a function of past observations. In our experiments, we use observation-independent behavior policies and hence omit importance sampling weights.

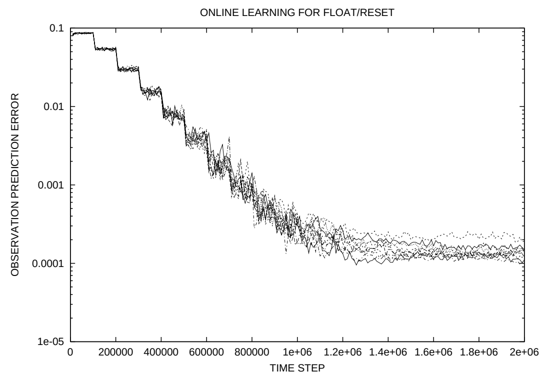

Figure 2. Prediction error for ten runs of our learning algorithm on float/reset.

4.1. Does it Work?

First 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 5. We used the core tests from 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 observation 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. It is also independent of the core tests being used and is useful for direct comparisons with existing methods based on other representations. Even though the true one-step observation probabilities are not generally available, in this case we can use our knowledge of the underlying system parameters to compute them solely for the purposes of measuring the error.

We computed the average error per time step over intervals of $10,000$ steps and plotted the sequence of errors for 10 runs in Figure 2 as learning curves. The step sizes for each run started at 0.1 and were 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. The error decreases to 0.0001 in each run, but does not go to zero in part because of the lower-bound clipping.

Comparison to History-Window and EM

For our second example, we highlight the advantage of PSRs over history-window and EM based approaches

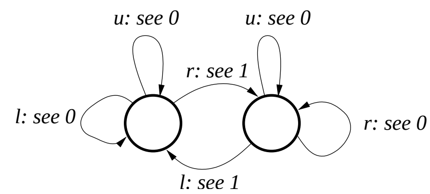

Figure 3. A simple two-state test system that no k-Markov model can represent.

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. Error is maximized for a 1-Markov model when the probability that u is chosen on any step is $1/2$. 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 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. 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 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 main purpose in presenting these comparative results is to demonstrate that our learning algorithm is indeed solving a non-trivial task.

POMDP Benchmarks

Emboldened by our success so far, we tested the performance of our learning algorithm on several of the problems from Cassandra's POMDP page (Cassandra, 1999). To determine the tests, we used the algorithm presented by Littman et al. (2002), which efficiently searches through the space of tests to find ones that produce linearly independent outcome vectors. We modified the search to be breadth first, which we have found produces shorter tests when there are multiple sets of tests that constitute a PSR. The majority of the tests were one step long on the problems we looked at, and no test was longer than two steps except in the Hallway problem. For each problem we ran our algorithm for ten million steps, starting with a step size of 0.1 and decreasing it by 10% every 100,000 steps down to a minimum of 0.00001. Results are presented in Table 1. Our error measure is the same one-step observation probability error we used for the float/reset problem in Section 4.1. Column 5 is the error produced when the behavior policy is uniformly random over the actions. Column 6 presents results with a behavior policy that chooses randomly from among the extension tests and executes it to completion before deciding again — this policy will be motivated at the end of the next subsection. Clearly our learning algorithm works better on some problems than others.

Problem	*Core tests*	*Act*	*Obs*	*Avg. error with policy:* *actions*	*ext. tests*
Tiger	2	3	2	0.000006	0.0000043
Paint	2	4	2	0.00001	0.000011
Cheese Maze	11	4	7	0.00037	0.00081
Network	7	4	2	0.001287	0.000826
Bridge Repair	5	12	5	0.00606	0.003389
Shuttle	7	3	5	0.026685	0.026661
4x3 Maze	10	4	6	0.066509	0.066399
Hallway	58	5	21	0.166763	0.196041

Table 1. Results for selected problems from Cassandra's POMDP page. The first column is the name of the problem. The next 3 columns denote the number of core tests used, the number of actions, and the number of observations. The last two columns show the performance of our learning algorithm for two different behavior policies.

Figure 4 shows the 4x3 maze (Russell & Norvig, 1994) problem from Table 1 on which our learning algorithm was less successful. It is a fairly straightforward grid-world in which movement is allowed in four directions, and the transitions are stochastic. Each action has an 80% chance of causing movement in the expected direction and a 10% chance of causing movement in either perpendicular direction. Moving into a wall has no effect. The observations indicate the presence of walls on the left side, right side, both sides, or neither side. The states labeled '+' and '-' are special goal states producing those observations, and taking any action in either of these states causes a transition to a different random state. We will use this 4x3 maze problem below in analyzing why our learning algorithm fails to find good models in some problems.

4.2. Further Analysis

This section analyzes various aspects of the performance of our learning algorithm.

Instability of Global Minimum

Here we ask the following question: how stable is the global minimum of the multi-step prediction er-

[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 successive iterations changed the log likelihood by less than 0.0001.

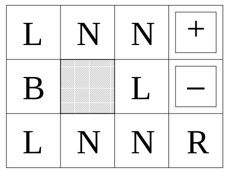

Figure 4. The 4x3 maze. Each state is labeled with the observation it produces.

Range	*Lowest error*	*Average error*	*Highest error*
.01	0.52383	0.632442	0.789186
.001	0.0963113	0.159012	0.256983
.0001	0.0101525	0.0156923	0.0232892
.00001	0.00167324	0.00372457	0.00671833
.000001	0.000195849	0.000537839	0.0011597

Table 2. Average error for ten runs of 4x3 maze with perturbed projections.

Range	*Lowest error*	*Average error*	*Highest error*
.01	1.11288e-004	2.88210e-004	4.89520e-004
.001	1.63735e-006	3.46128e-006	6.33837e-006
.0001	2.73582e-008	4.73308e-008	1.09015e-007
.00001	1.67552e-010	5.77677e-010	1.48383e-009
.000001	2.81815e-012	4.84912e-012	1.02960e-011

Table 3. Average error for ten runs of float/reset with perturbed projections.

ror function (Equation 4) we used to derived our algorithm? As a first experiment in the 4x3 maze, we used the true prediction vectors at each time step and updated our projection (parameter) vectors normally using our learning algorithm. That is, during learning we used our knowledge of the underlying system to calculate the true prediction vectors, instead of calculating them using our estimated projection vectors as we would normally do. Doing so led to very low errors, confirming that at least when the myopic assumption made by our learning algorithm is met, it learns well. Next, we did an experiment in which we started the projection vectors at the global minimum by setting them to analytically derived optimal values and then updated them normally (i.e., with estimated projection vectors) using our learning algorithm. Even with a small step size of 0.00001, the projection vectors drift away from the global minimum and the error rises significantly. This behavior shows an instability around even the global minimum that presents a significant problem for our learning algorithm.

To empirically confirm that at least the *expected* myopic gradient at the global minimum of the error function is zero, we ran the same experiment as above except in batch-mode wherein we collected the changes proposed in the projection vectors instead of actually making them. The average change in the projection vectors converges to zero as we increase the size of the batch confirming that the expected myopic gradient is indeed zero. The online and batch results taken together seem to imply that small errors in the projection vectors caused by the noise in the stochastic myopic gradient can feed into the errors in the prediction vectors which can feed back into the errors in the projection vectors and so on to make the updates unstable around error-function minima.

Sensitivity to Parameters

To get a picture of the error surface near the global minimum mentioned above, we experimented with slightly perturbing the correct projection vectors and observing the resulting error. To each component of each correct projection vector, we added a uniformly random perturbation amount in the range $[-x, x]$ for values of x between 0.01 and 0.000001. We then updated the prediction vector for 100,000 steps without updating the weights, and recorded the average error over all steps. The results are presented in Table 2. For comparison, Table 3 gives the results of the same experiment on float/reset. The error increases rapidly as the weights move away from the solution to the 4x3 maze, while the error surface around the solution to float/reset is much more shallow.

While these relative sensitivity results for float/reset and 4x3 maze do not by themselves explain the differing performance of our learning algorithm on these two problems, they do suggest a correlation between qualitative aspects of the error surface and the performance of our learning algorithm.

Robustness to Extra or Fewer Tests

Here we consider the effect of having extra core tests or too few core tests on the performance of our learning algorithm for a problem on which it does well, namely the float/reset problem.

Extra Core Tests: We can add to the core tests used for float/reset in Section 4.1 by adding extra $f0$ pairs before the longest core test $f0f0f0r0$. Gradually increasing the number of core tests in this fashion does not appreciably change the final error, though it does increase the variance in this error. Figure 5 shows the result of including core tests of this form up to and including eight float actions before the reset.

Missing core tests: What if we don't include some

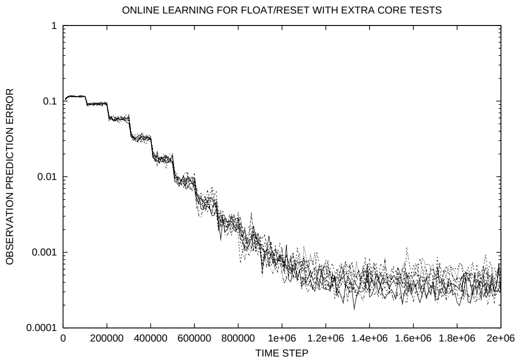

Figure 5. These trials included extra core tests that took the form of n $f0$ tests followed by an $r0$, for $4 \leq n \leq 8$.

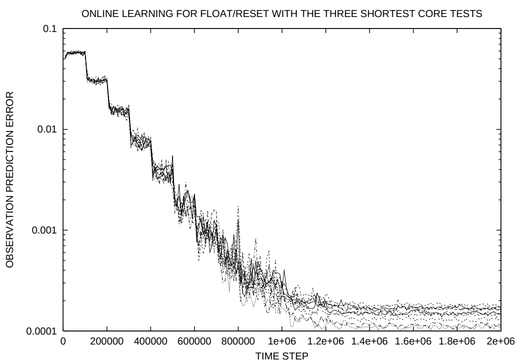

Figure 6. These trials only included three core tests: $f0$, $r0$, and $f0r0$. The "steps" correspond to learning-rate changes.

of the 5 core tests used for float/reset in Section 4.1? In such a case, the reduced set of tests will not be a PSR. Figure 6 shows that our learning algorithm achieves the same level of performance with only the three shortest core tests, $f0$, $r0$, and $f0r0$, as it does with the original five. Further experimentation reveals that sets of core tests that include these three achieve error close to 0.0001; while any set that omits any one of these three core tests achieves no better than ten times that error.

While the effect of extra or missing core tests is likely to be domain dependent, the fact that our results on float/reset are robust to changes in the core tests is encouraging with regards to the possibility of eventually combining core-test discovery with learning. On the other hand, the fact that one needs certain subsets of tests to get significant reduction in prediction error suggests that discovering the core tests by some incremental generate and test approach may be difficult.

Behavior Policy Matters

We also performed a separate series of experiments with our learning algorithm in simple scalable gridworlds with 4 actions (move N,S,E,W) and two observations (1 if an action would cause a move off the edge of the grid, and 0 otherwise). For lack of space we will not present many of the experimental details here but simply report a finding. Note that a result of using a uniform random policy is that the probability of the execution of a test goes down exponentially with its length. This unbalanced experience with tests of different lengths prevented our learning algorithm from learning effectively in grid worlds. We tried a different behavior policy that chose randomly among extension tests and executed their action sequences to completion before making another choice. This uniform extension-test behavior policy balances experience with the different tests more evenly and allowed our learning algorithm to learn in the same grid worlds that it was having trouble with earlier.

5. Discussion

Our learning algorithm uses the simplest approximation to the gradient of the multi-step error function. Our empirical results show that, despite this fact, it is able to learn a good model for some dynamical systems. Our analysis of the cases where it fails has already led to our making changes in the learning algorithm that have improved its performance. For example, the discovery that long core tests occur infrequently and hence don't get as many updates and thereby contribute significantly to the error led us to two changes: 1) to use shorter core tests (by using breadth-first search), and 2) to use a behavior policy that explores tests rather than primitive actions. In addition, our analyses of how stable the global minima are and how sensitive the prediction error is to slight changes in the parameters lead us to believe that the difficulty our learning algorithm had in some domains is explained in part by the different error landscapes in different systems. For example, the global minimum seems to be more stable and less sensitive for the float/reset problem (which was easy to learn) than for the 4x3 maze (which was hard to learn). We speculate that this difference will be encountered with most gradient-based learning algorithms. In addition, there are still other potential ways in which the performance of our learning algorithm could be improved. For instance, it is possible that with other heuristics commonly used with gradient algorithms like random restarts or the use of a momentum term, our learning algorithm's performance can be further improved. Regardless, our learning algorithm is an important first step towards the development of robust algorithms for learning with PSRs. In our future work, we intend to explore both less-myopic gradient-based algorithms as well as non-gradient algorithms.

Appendix

We show that our learning algorithm is a myopic version of the gradient algorithm for the following multi-step prediction error function:

$$\begin{aligned} \text{Error(t)} &= \sum_{x \in X_{|a_t}} [p(x|h_{t-1}) - \hat{p}(x|h_{t-1})]^2 \\ &= \sum_{x \in X_{|a_t}} [p(x|h_{t-1}) - \hat{p}^T(Q|h_{t-1})\hat{m}_x]^2, \qquad (6) \end{aligned}$$

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 $x \in X_{|a_t}$

$$\begin{aligned} \frac{\partial \text{Error}(t)}{\partial m_x} &= -\Big(2[p(x|h_{t-1}) - \hat{p}^T(Q|h_{t-1})\hat{m}_x] \\ &\quad [\hat{p}(Q|h_{t-1}) + \frac{\partial \hat{p}(Q|h_{t-1})}{\partial m_x}\hat{m}_x]\Big), \qquad (7) \end{aligned}$$

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

$$\begin{aligned} \frac{\partial \text{Error}(t)}{\partial m_x} &= -\Big(2[p(x|h_{t-1}) - \hat{p}^T(Q|h_{t-1})\hat{m}_x] \\ &\quad \frac{\partial \hat{p}(Q|h_{t-1})}{\partial m_x}\hat{m}_x\Big). \qquad (8) \end{aligned}$$

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 $x \in X$

$$\begin{aligned} D_x(t) &= -\chi_{x \in X_{|a_t}}\Big(2[p(x|h_{t-1}) - \hat{p}^T(Q|h_{t-1})\hat{m}_x] \\ &\quad \hat{p}(Q|h_{t-1})\Big), \qquad (9) \end{aligned}$$

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

$$\begin{aligned} d_x(t) &= -\chi_{x \in X_{|a_t}}\frac{1}{w_{x,t}}\Big(2[\chi_{x,t} - \hat{p}^T(Q|h_{t-1})\hat{m}_x] \\ &\quad \hat{p}(Q|h_{t-1})\Big) \qquad (10) \end{aligned}$$

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 5 is to follow the myopic approximation of the multi-step prediction error gradient, in expectation.

Acknowledgments

We are grateful to Richard Sutton for numerous inspiring interactions on PSRs and for many specific ideas and suggestions on what would make a good algorithm learning PSRs. Natalia Hernandez Gardiol and Daniel Nikovski provided pointers to relevant materials.

References

Cassandra, A. (1999). Tony's pomdp page. `http://www.cs.brown.edu/research/ai/pomdp/index.html`.

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* (pp. 1555–1561).

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.

Russell, S. J., & Norvig, P. (1994). *Artificial intelligence: A modern approach.* Englewood Cliffs, NJ: Prentice-Hall.