Spectral Learning in Sparse Environment

1 Predictive state representations

As it was first explained by [Littman 2002], PSR models a dynamic system via observable quantities. It represents the state of an environment as a prediction vector over the observable experiments called tests set. In essence, a test of length k is a sequence of actions and observations like $t = a_1o_1a_2o_2...a_ko_k$. We call t succeeded if given actions $a_1...a_k$, we observe $o_1...o_k$. In a similar way, we can define a history as a sequence of actions and observations that has occurred in the past usually before a test. In PSR, actions are executed by some policy and observation is the response of the dynamic system. In PSR, the main idea behind the scene is that if we know the expected result of executing all possible tests, we have sufficient information about the state of the environment. In this regard, a system dynamic matrix D is a bi-infinite matrix which its rows corresponds to history and its columns corresponds to tests. Each entry $[i, j]$ of this matrix is defined as $D_{ij} = p(t_j|h_i)$. The rank of this matrix corresponds to the dimension of our dynamical system . As a result, a set of linearly independent columns of D provide sufficient statistics to predict other tests given any history h. We call these tests our core tests denoted by $Q = \{q_i\}_{i=1}^{K}$. Following Boots et al., 2011], we can write the prediction vector over our core test set Q given a history h as:

$$p(Q|h) = [p(q_1|h), ..., p(q_K|h)]^T \tag{1}$$

In general, for a given test t, we can find a vector m_t such that $p(t|h) = m_t^T p(Q|h)$. It was first suggested by [Singh et al., 2004] that the idea can be expanded to include non-linear function as well. In this case, we can say $p(t|h) = f_t(p(Q|h))$ for some non-linear function f_t. For the rest of this paper we will focus on linear PSR and thus will continue referring to them as PSR. As $Q(h)$ provides sufficient statistics to compute the probability of other tests, in PSR we call it the state of our environment after the observed history h.

In order to update $Q(h)$ after executing action a and seeing observation o, we define matrix M_{ao} whose j^{th} column is the vector m_{aoq_j} and we have:

$$p(Q|h) = [p(q_1|h), ..., p(q_K|h)]^T \tag{2}$$

$$p(Q|hao) = \frac{M_{ao}^T p(Q|h)}{m_{ao}^T p(Q|h)} = \frac{M_{ao}^T p(Q|h)}{m_\infty^T M_{ao}^T p(Q|h)} \tag{3}$$

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

where m_∞ is a normalizer vector such that $m_\infty^T Q(h) = 1 \ \forall h$.

Formally, we can express PSR as a tuple, $\{O, A, Q, F, m_0\}$ such that O is the observation set, A is a set containing possible actions, Q are defined as previously, F is the set of prediction functions (vectors) and m_0 is the initial state. By using previous terminology, we can say $m_0 = Q(\epsilon)$ where ϵ refers to empty history and F contains vectors m_t.

TPSR models are a natural generalization of PSR models. In TPSR instead of maintaining the actual probability distribution over the core tests, it uses a linear combination of probability of a set of tests as a sufficient statistics. In fact, TPSR can be seen as a linear transformation of PSR models. As shown by [Boots et al., 2011], by using TPSR, parameter learning can be done in a closed form. In this regard, people usually use SVD decomposition to tune the complexity and parameter learning of TPSR. In this work, we have provided a novel algorithm which is used instead of the SVD decomposition and results in a better accuracy of TPSR prediction.

1.1 Parameter computation

Here, we first describe the observable quantities related to PSR model and then based on that we construct the quantities corresponding to TPSR. Consider core tests set T and a statistically sufficient set H of indicative events. H contains mutually exclusive and exhaustive partition of the set of histories. Each partition in H is denoted by H_i which is a possible histories. We can now define vectors and matrices in terms of observable quantities. $P_H \in R^{|H|}$ is a vector defined as $P_H = P(H)$, whose elements are $P(h \in H_i)$ with h sampled according to some distribution ω. Next, consider a matrix $P_{H,T} \in R^{|H|\times|T|}$ whose entries corresponds to the joint probability of test and histories. We assume that test happens after the history. In this case, for each $[i, j]$ entry of $P_{H,T}$ we have:

$$\begin{aligned}(P_{H,T})_{i,j} = P(H_i, t_j) =& P(t_j|H_i)P(H_i) \\ =& E(Q(h)^T m_{t_j}|H_i)P(H_i) \\ =& P(H_i)s_{H_i}^T m_{t_j}\end{aligned} \tag{4}$$

where $H_i \in H$ and $t_j \in T$ with $h \sim \omega$. In (4) s_{H_i} is a compact notation for $E(p(Q|h))$. It is easy to check that we can write the whole $P_{H,T}$ in the following form:

$$P_{H,T} = DSM \tag{5}$$

where $M \in R^{|Q|\times|T|}$ is a matrix whose columns are the vector m_{t_j}. $S \in R^{|H|\times|Q|}$ is a matrix with rows s_{H_j} and $D = diag(P_{H_i})$.

Next, we can also observe one step (action-observation pair) extension of the matrix $P_{H,T}$. It is called $P_{H,ao,T}$ and can be defined for each possible action-observation pair. This new matrix contains the joint probability of a history plus action-observation pair plus a test. Like before, we can say:

$$\begin{aligned}(P_{H,ao,T})_{i,j} = P(H_i, ao, t_j) =& E(P(t_j, ao|h)|H_i)P(H_i) \\ =& E(P(t_j|ao, h)P(ao|h)|H_i)P(H_i) \\ =& E(p(Q|hao)^T m_{t_j} P(ao|h)|H_i)P(H_i) \\ =& E(p(Q|h)^T M_{ao} m_{t_j}|H_i)P(H_i) \\ =& P(H_i)s_{H_i}^T M_{ao} m_{t_j}\end{aligned} \tag{6}$$

In general, it is easy to check that we have:

$$P_{H,ao,T} = DSM_{ao}M \tag{7}$$

Finally, the last observable parameters needed to be calculated is m_0 and m_∞. If the dynamical system does not have a reset function, we cannot estimate m_0. Instead due to the Markov property, we can consider an arbitrary state like $m_* = S^T P(H)$ to be used in place of m_0 in our computation. Note that for m_∞, we can easily show that $m_\infty^T = 1_{|H|}^T (S^T)^\dagger$ [Boots et al., 2011].

For finding the parameters of TPSR, we know that TPSR is just a linear transformation of PSR models. As discussed earlier, to compute TPSR, usually the SVD of $P_{H,T}$ is taken and then the right singular vectors $V \in R^{r\times|Q|}$ is used for the linear transformation. r is the rank of the matrix $P_{H,T}$.

In some papers, left singular vector U might also be used to do this transformation; however, this minor change is not going to affect the generality of our argument here.

The parameters of TPSR is computed bellow:

$$\begin{aligned} b_* = (V^T M^T) m_* =& (V^T M^T)(S^T P(H)) \\ =& (V^T M^T)(S^T D) 1_{|H|} \\ =& (DSMV)^T 1_{|H|} \\ =& (P_{H,T} V)^T 1_{|H|} \end{aligned} \tag{8}$$

$$\begin{aligned} b_\infty^T = m_\infty^T ((MV)^T)^{-1} =& 1_{|H|}^T (S^T)^\dagger (V^T M^T)^{-1} \\ =& 1_{|H|}^T (S^T)^\dagger (V^T M^T)^{-1} (V^T M^T)(S^T D)((DSMV)^T)^\dagger \\ =& 1_{|H|}^T (S^T)^\dagger (S^T D)((P_{H,T} V)^T)^\dagger \\ =& 1_{|H|}^T (D)((P_{H,T} V)^T)^\dagger \\ =& ((P_{H,T} V)^T)^\dagger \end{aligned} \tag{9}$$

$$\begin{aligned} B_{ao} = (MV)^T M_{ao}^T (MV)^{-T} =& (MV)^T M_{ao}^T (MV)^{-T} (MV)^T (S^T D)(V^T M^T S^T D)^\dagger \\ =& (MV)^T M_{ao}^T (S^T D)((DSMV)^T)^\dagger \\ =& V^T (M^T M_{ao}^T S^T D)(V^T P_{H,T}^T)^\dagger \\ =& V^T (DSM_{ao} M)^T (V^T P_{H,T}^T)^\dagger \\ =& V^T (P_{H,ao,T})^T (V^T P_{H,T}^T)^\dagger \\ =& ((P_{H,T} V)^\dagger P_{H,ao,T} V)^T \end{aligned} \tag{10}$$

By using the above parameters, we can calculate the probability of a test containing a sequence of action-observations pair as follows (assuming that the system has started from state m_*):

$$\begin{aligned} p(a_1 o_1 ... a_n o_n) =& p(o_1 ... o_n | a_1 ... a_n) \\ =& m_*^T M_{ao_{1:n}} m_\infty \\ =& m_*^T (MV)(MV)^{-1} M_{ao_{1:n}} (MV)(MV)^{-1} m_\infty \\ =& b_*^T B_{ao_{1:n}}^T b_\infty \end{aligned} \tag{11}$$

2 Algorithm

Before trying to propose an algorithm to refine the computation of the TPSR model, let's take a closer look to the meaning of taking SVD of the matrix $P_{H,T}$

Theoreom 1[Watkins, 1998] Let $A \in R^{nxm}$ be a matrix with rank$(A) = r > 0$ and the SVD of A be $U \Sigma V^T$. Let $A_k = U \Sigma_k V^T$, where Σ_k contains just the first k singular values on the diagonal of the matrix Σ and its other diagonal entries are zero. This implies rank$(A_k) = k$ and we have (F stands for frobenius norm):

$$A_k = \underset{rank(B) \leq k}{\arg\min} ||A - B||_F^2 = \underset{rank(B) \leq k}{\arg\min} \sum_{i,j} (A_{i,j} - B_{i,j})^2 \tag{12}$$

As a result of Theorem 1 when we are using the right singular vectors of $P_{H,t}$ for transformation, we are using the solution of the following optimization problem:

$$min_{rank(B) \leq k} \sum_{i,j} ((P_{H,T})_{i,j} - B_{i,j})^2 \tag{13}$$

Where we have assumed the rank of the matrix $P_{H,t}$ is k. One problem with (13) is that it is assigning equal weight to all entries of the matrix $P_{H,T}$. In practice; however, we just have an estimate of this

matrix built upon training data. It is possible in the training phase, the samples might be biased due to the cost of gathering special data or any other reasons. This implies that the accuracy of some entries of $P_{H,T}$ might be higher than the others which makes using the right singular vectors as a transformation matrix questionable. Here we try to propose a better algorithm for finding a matrix to be used instead of V. Consider two matrices $Q \in R^{|H|\times|k|}$ and $R \in R^{|k|\times|T|}$. Instead of using (13) for finding the transformation we use the following objective to find matrices Q, R:

$$min \sum_{i,j} w_{i,j}((P_{H,T})_{i,j} - Q_i R_j)^2 = min \sum_{i,j} ||P_{H,T} - QR||^2_{F,W} \tag{14}$$

where $w_{i,j}$ denotes the weight of the entry $(P_{H,T})_{i,j}$ which is equal to the number of samples we used to estimate it. W is a matrix containing $w_{i,j}$ as its elements. In (14) Q_i corresponds to the ith row of the matrix Q and R_j is the jth column of the matrix R. Note that the condition $rank(QR) \leq k$ is not included here as it is already satisfied due to the definition of the matrices Q, R. We can use gradient descent to solve (14) to find Q, R. Moreover, the matrix R has similar structure to the matrix V we already used in our parameter calculation except with R we have also considered the weights. It seems that R has a clear advantage and we use it instead of V in the TPSR parameters. Although in some cases using (14) would lead to better prediction, it does not guarantee that QR matrix keeps the structure of $P_{H,T}$. In what follows by $h_i t_j$ we mean the concatenation of the two action observation pairs h_i and t_j respectively. The main properties that $P_{H,T}$ has are:

- Its entries lie in $[0, 1]$ interval.
- If for any two pairs (h_1, t_1) and (h_2, t_2), we have $h_1 t_1 = h_2 t_2$ then we have $P_{H,T}(h_1, t_1) = P_{H,T}(h_2, t_2)$
- $\sum_{o\in O} P_{H,T}(h, tao) = P_{H,T}(h, t) \ \ \forall a \in A$, where A, O are possible actions and observations set.

In our experiments, the first property was rarely violated in QR multiplication, so we ignored it. We are now going to add some additional term to our initial objective (14) to force the second and third property in QR calculation. Let:

$$C = \{C_i| \ \forall (h_1, t_1), (h_2, t_2) \in C_i : h_1 t_1 = h_2 t_2 \ \& \ \neg\exists (h_1, t_1) \in C_i, (h_2, t_2) \notin C_i : h_1 t_1 = h_2 t_2\}$$

So, C consists of sublists C_i containing all the tuples that their probability should be the same in $P_{T,H}$. Also, here we assume that these sublists are distinct. For the third property, note that it is actually a sum over columns of $P_{H,T}$, so we need to enforce it on R no matter what the Q values are. Define Y such that:

$$\sum_{y\in Y} (P_{H,T})_{:,y} - (P_{H,T})_{:j} = 0$$

Now, by using the set Y and the set C we can add the second and third property by using regularization:

$$min \sum_{i,j} ||P_{H,T} - QR||^2_{F,W} + \lambda_1 (\sum_{k\in|C|} \sum_{i,j\in C_k} (q^T_{x_i} r_{y_i} - q^T_{x_j} r_{y_j})^2) + \lambda_2 \sum_{j=1}^{n} (\sum_{y\in Y} R_{:,y} - R_{:,j})^2 \tag{15}$$

where x_i, y_j denotes the row and column number of element i in $P_{H,T}$. The whole algorithm is written bellow. We call this algorithm denoising algorithm

Data: $A \in R^{m \times n}$, step size α , termination threshold ϵ, regularization parameters λ_1, λ_2
Result: Rank-k matrix M
A=QR;
while $\Delta(||A - QR||^2_{F,W} + \lambda_1(\sum_{k\in|C|}\sum_{i,j\in C_k}(q_{x_i}^T r_{y_i} - q_{x_j}^T r_{y_j})^2)$
$+\lambda_2 \sum_{j=1}^{n}(\sum_{y\in Y} R_{:,y} - R_{:,j})^2) > \epsilon$ **do**
  for i *in* $range(0, m)$ **do**
    for j *in* $range(0, k)$ **do**
      $Q_{ij} = Q_{ij} + \alpha\lambda_1(\sum_{t=1}^{n} 2R_{jt}(A_{it} - q_s^T r_t))))$;
      if $i == x_t \in C_k$ **then**
        $Q_{ij} = Q_{ij} - \alpha\lambda_1(\sum_{k\in|C|}\sum_{s,t\in C_k} -2R_{j,y_t}(q_{x_s}^T r_{y_s} - q_i^T r_{y_t}))$
      else if $i == x_s \in C_k$ **then**
        $Q_{ij} = Q_{ij} - \alpha\lambda_1(\sum_{k\in|C|}\sum_{s,t\in C_k} 2R_{j,y_t}(q_{x_s}^T r_{y_s} - q_i^T r_{y_t}))$
    end
  end
  for i *in* $range(0, k)$ **do**
    for j *in* $range(0, n)$ **do**
      $R_{ij} = R_{ij} + \alpha(\sum_{s=1}^{m} 2Q_{si}(A_{sj} - q_s^T r_t))))$;
      if $j == y_t \in C_k$ **then**
        $R_{ij} = R_{ij} - \alpha\lambda_1(\sum_{k\in|C|}\sum_{s,t\in C_k} -2Q_{x_t,i}(q_{x_s}^T r_{y_s} - q_{x_t}^T r_j))$
      else if $j == y_s \in C_k$ **then**
        $R_{ij} = R_{ij} - \alpha\lambda_1(\sum_{k\in|C|}\sum_{s,t\in C_k} 2Q_{x_s,i}(q_{x_s}^T r_j - q_{x_t}^T r_{y_t}))$
      if $j \in Y$ **then**
        $R_{ij} = R_{ij} - 2\alpha\lambda_2(\sum_{y\in Y} R_{iy} - R_{iz}$
      end
      Let t = col(j)
      if $\exists\ ao\ s.t.\ ind(tao) \in Y$ **then**
        $R_{ij} = R_{ij} + 2\alpha\lambda_2(\sum_{y\in Y} R_{iy} - R_{ij}$
      end
    end
  end
end

Algorithm 1: Denoising Algorithm

3 Experiment

To show the capacity of our proposed denoising algorithm, we conducted several experiments.

3.1 Simple One State Environment

In this experiment we considered an environment having one state, two actions and two observations. The policy we are using here is a simple policy: the agent begins in policy state A. It chooses the action right with probability $1/k$ and the action left with probability $1 - 1/k$. It then transitions to policy state B, where it always chooses to take the action left. It then transitions back to state A, and so on. If the agent takes the left action, then the observations 0 and 1 will be output with equal probability $(1/2)$. If the action right is taken, then observation 0 is always obtained. Thus, in this example, we have A = {left,right} and O = {0, 1}. The environment and the policy figures are depicted in Figure 1. The results for k=100 has been depicted in Figure 2 and Figure 3 for different training size and the error bar is the 1-norm. As our core tests, we included all the length 2 possible tests and for the histories we considered all the length 1 histories. As it can be seen, in both figures the denoising algorithm is outperforming the nondenoising one.

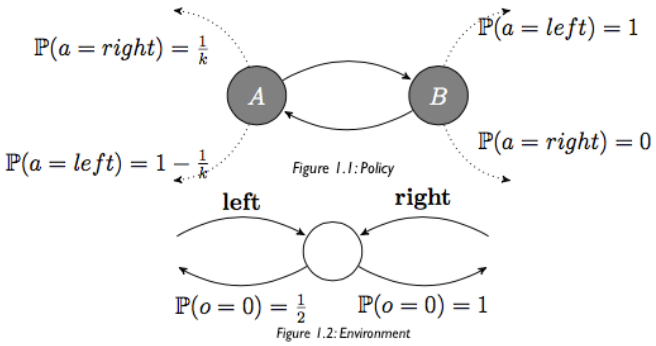


Figure 1: One State Environment.

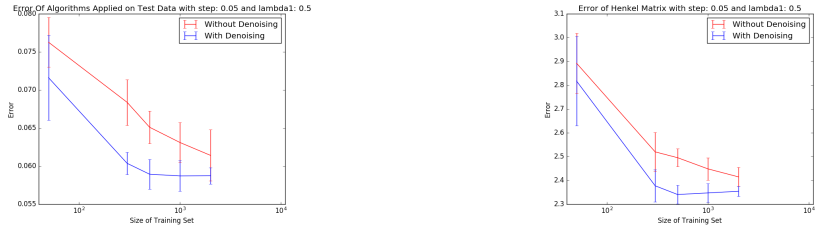
(a) Error On Test Data

(b) Error in Henkel Matrix

Figure 2: The result of applying denoising algorithm. In this experiment the rank of Henkel matrix is considered to be 1

3.2 Torus Environment

In this experiment we tried to create a simulated environment that might happen in a real environment. Assume a robot exploring in a maze where it can take any actions at any point except for specific time that happens periodicly. In such time the robot cannot take specific actions. In essence, this property makes this kind of environment biased. For example, a robot that is exploring a mountain cannot take specific actions at certain time as those actions might cause death fall. Motivated by these observations, we created a torus environment where a robot tries to explore it. As it can be seen in Figure 4 (a), the white dots on the torus corresponds to the locations that robot can go, the red line on the boundary of the torus represents the paths that robot can follow via its policy to change its location. As it can be seen, on each horizontal path, the robot can move left, right, up and down only on specific locations that are apart from each other by three spaces. So, the environment is biased. The observation is the color of the patch beneath the robot. Here we have assumed four observation and eight actions. On the intersections, the robot can take 8 actions to move to its both vertical and horizontal neighbours locations and in other cases it can take only 4 action which moves robot vertically. The observation distribution of seeing a colour is different in each location and we



(a) Error On Test Data

(b) Error in Henkel Matrix

Figure 3: The result of applying denoising algorithm. In this experiment the rank of Henkel matrix is considered to be 2

(a) (b)

Figure 4: The torus environment (a). Figure (b) corresponds to the exploration of robot (yellow line) and different observation it might perceive(the color of patches).

(a) Error on test data (b) Error of Henkel matrix

Figure 5: The result of applying denoising algorithm on torus environment

used Dirichlet distribution to assign these values. The robot chooses any valid action with almost equal probability. The figures of applying denoising algorithm is depicted in Figure 5. We have used rank 15 as we got better predictions for both usual and denoising method.

3.3 Torus Expanded Environment

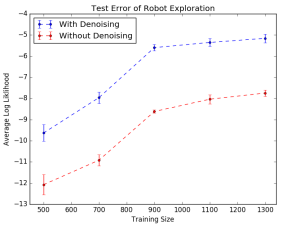
After getting good result on a limited number of observation, we expanded the torus environment up to 20 possible observation in each cell. We used dirichlet distribution for assigning different probability to observation of each cell. One drawback of expanding observation to 20 was that the overall probability of a trajectory would decrease dramatically. In order to avoid that for each 5 consecutive states appearing in each horizontal line of torus border, we partitioned randomly 20 observation to 5 sets of 4 observation and assigned one of these set to a state so that the state had zero probability on all observation except for the 4 observation in this set. At the end these 5 states covers all the 20 observations. Figure 6 shows the result of applying denoising versus non denoising. Our test data was 5000 trajectories up to length 6.

3.4 Robot

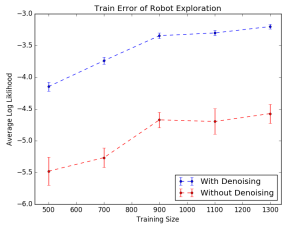
After getting good results on simulated environments, we tried to see the effect of denoising algorithm when applied on a real dataset. Here, we have used the dataset gathered by a robot navigation. The

(a) Error on test data (b) Error of Henkel matrix

Figure 6: The result of applying denoising algorithm on expanded torus environment



(a) Test error of 100 observations



(b) Train error of 100 observations

Figure 7: Robot Experiment

dataset was gathered in 2009 for the purpose of comparing different neural network performance. The robot had 24 scanners which collected the data while exploring the environment. As the gathered data was continuous, we made it discrete by sampling a number of observation and then relabelling all the observation by giving the label of nearest sampled observation in 2-norm. We then splitted the whole dataset into training and testing set and did 5 fold cross validation. We selected 100 observation (Figure 7). As it can be seen, the denoising algorithm has a great effect on improving the prediction of TPSR model. We used the rank 40 based on the singular values of Henkel matrix. The running time of Denoising algorithm for 100 observation was 1 hour and 30 minutes with 4 core CPU system.