Neural Network Based Nonlinear Predictive State Representation

Tianyu Li
School of Computer Science
McGill University
Montreal, Quebec H3A 2A7
Email: tianyu.li@mail.mcgill.ca

***Abstract*—Predictive state representation (PSR) can expressively model dynamical system. However, a PSR is inherently a linear model. In this report, we propose a neural network based nonlinear PSR model along with a learning algorithm. Our learning algorithm performs a *nonlinear* decomposition of the so-called history test probability matrix (using an encode-decoder neural network) from which the prediction functions of the model are recovered. We assessed the performance of the proposed model in a simulation study.**

1. Preliminaries

We first introduce notions on predictive state representation and the transformed PSR.

Predictive state representation. To introduce predictive state representation(PSR) formally, we need to introduce the concept of *history* and *test*. Let us now consider dynamical systems that accept actions from a discrete set A, and generate observations from a discrete set O. At time step k, a test t is defined as the action observation pairs starting from time step t, i.e. $t = a_{k+1}o_{k+1}a_{k+2}o_{k+2}\cdots a_{k+n}o_{k+n} \in T$; while a history h is defined as the action observation pairs before time k, i.e. $h = a_1o_1a_2o_2\cdots a_{k-1}o_{k-1} \in H$. Among these tests, there exists a set of core tests $Q_t = \{q_1, q_2, \cdots, q_u\}$, for which the conditional probability given history h, defined as $Q(h) = [P(q_1|h), P(q_2|h), \cdots, P(q_k|h)]$ is the sufficient statistics for predicting the future, i.e. $P(t|h) = f_t(Q(h))$, which we will refer $f_t(\cdot)$ to prediction function.

As we seeing more action observation pair ao, we need to update our sufficient statistics $Q(h)$, let us now assume we have linear prediction functions, i.e. $f_t(x) = mx$, where m is a vector. For the i-th component of $Q(hao)$: $P(q_i|hao)$, we have the following equations:

$$\begin{aligned} P(q_i|hao) &= \frac{P(aoq_i|h)}{Pao|h} \\ &= \frac{f_{aoq_i}(Q(h))}{f_{ao}(Q(h))} \\ &= \frac{m_{aoq_i}Q(h)}{m_{ap}Q(h))} \end{aligned} \tag{1}$$

The objective for now is just to predict the system instead of controlling it. Let Q define a set of core tests, m_0, be the initial vector of empty history λ, F be the set of prediction functions, then a predictive state representation is defined as a tuple $< A, O, Q, m_0, F >$. Littman et. al. [Littman et al.] showed that compared to POMDP, PSR requires no more than the states needed for the corresponding POMDP, and given the colsed form solution, PSR became a very great tool for modeling the system dynamics.

Transformed PSR. Transformed PSR are a variant of PSR. TPSR are capable of tuning the complexity of their representation to match the specific problem being addressed [Rosencrantz et al., 2004]. The key difference between PSR and TPSR is that instead of maintaining the sufficient statistics directly, TPSR maintains a linear combination of the sufficient statistics.

Let us now introduce the observable matrices. The first matrices is the history probability $[P_H]_i = P(h_i), h_i \in H$. Next is the joint distribution matrix of history and test. For a history h_j and a test t_i, the i, j element of the matrix is $[P_{H,T}]_{i,j} = P(h_j, t_i)$, it has been shown in [James and Singh, 2004], $[P_{H,T}]_{i,j} = DSR$, where $S = E(Q(h)|h_j), D = diagP_H, R = M_{t_j}$. The third one is $P_{H,ao,T} \in \mathcal{R}^{|H|\times|T|}$. Similar factorization also has been shown in [James and Singh, 2004], $[P_{H,ao,T}]_{i,j} = P(h_j, ao, t_i) = DSM_{ao}R$.

The corresponding spectral learning method is actually by taking advantages of these matrices factorizations. By taking the right singular matrix V of $P_{H,T}$, TPSR takes the form of:

$$b^* = 1_* P_{H,T} V^T \tag{2}$$

$$b_\infty = P_H (V_T P_{H,T}^T)^+ \tag{3}$$

$$B_a o = (V_T P_{H,T})^+ P_{H,ao,T} V^T \tag{4}$$

where 1_* is all zeros vector except the *th element being 1, b_* defines the initial state vector staring from any arbitrary history h_*, b_∞ is a normalization vector, it can be interpreted as certain termination condition.

The new update rule can be then written as following:

$$b_{t+1} = \frac{B_{aot} b_t}{b_\infty^T B_{aot} b_t} \tag{5}$$

2. Introducing Non-linearity

In this section we will discuss about the possibility of introducing non-linearity into the original linear PSR and we will then provide our learning algorithm.

Where to apply non-linearity. Consider the TPSR we just introduced, for a TPSR, the goal is to firstly find a linear combination of the sufficient statistics provided by the core test, and then try to solve the linear prediction function f_t so that one will obtain the update rule. Therefore, form here, we can actually see there are two aspects of applying non-linearity, one is to find a nonlinear representation of the sufficient statistics and the other is to use nonlinear prediction function. By doing so, we are hoping the introduced non-linearity can help PSR to describe wider range of dynamical system, i.e. nonlinear system, and potentially reduce the size of the model(number of hybrid of sufficient statistics).

How to apply non-linearity. Let us reconsider the factorization mentioned above. Define a nonlinear function $\Phi : R^{|Q|} \rightarrow R^{|Q_{new}|}$, where Q_{new} is the sufficient statistics after nonlinear functions combination, and we also assume the prediction functions f_t now is nonlinear. For the history test matrix $P_{H,T}$, we have:

$$\begin{aligned} P_{h_j,t_i} &= P(t_i|h_j)P(h_j) \\ &= f_{t_i}(\Phi(Q(h_j)))P(h_j) \end{aligned} \tag{6}$$

Therefore we have $P_{H,T} = f_s(f_p(P_H))$, where $f_p(\cdot) = \Phi(Q(\cdot))$ and $f_s(\cdot) = f_t(\cdot)P_H$, as our new nonlinear factorization. Here we can see f_p controls the embedding of the original sufficient statistics in the nonlinear latent space while f_s involves nonlinear prediction function. However, we want to solve the prediction function, and this can be done by the similar procedure of the third matrix.

$$\begin{aligned} P_{h_j,ao,t_i} &= P(t_i|h_j ao)P(ao|h_j)P(h_j) \\ &= f_{t_i}(f_{ao}(\Phi(Q(h_j))))P(h_j) \end{aligned} \tag{7}$$

The matrix form can be written as $P_{H,ao,T} = f_s(f_{ao}(f_p(P_H)))$. Hence, after obtaining the previous f_p and f_s, we will be able to learn f_{ao}. Now, we need to give an exact form of these two nonlinear functions. To introduce non-linearity, people often choose kernel based method, or go with neural nets. For our case, it seems straightforward to apply neural networks in our model.

Learning algorithm. Inspired by the matrices factorizations in TPSR, we propose our neural network based algorithm for learning nonlinear PSR. This framework can both work with nonlinear combination of sufficient statistics and nonlinear prediction functions.

Let us refer the factorization of $P_{H,T}$ to factorization step, while the factorization of $P_{H,ao,T}$ to regression step. For the factorization step, the proposed neural networks architecture is shown in Figure 1:

Here we have P-net correspond to f_p, while S-net correspond to f_s. This network takes randomly generated samples x, and try to minimize the error towards $xP_{H,T}$. After we obtaining the factorization nets, we save the weights of the corresponding layers and then move on the solve the regression ste, as shown in Figure 2: Here the input is still randomly generated, while the objective is to minimize the prediction error towards $xP_{H,ao,T}$. After training this net, we extract the middle TranNet, based on different combination of ao, we have obtained our f_ao. As for the normalization factor b_∞, we set it to be another function $f_{term}(\cdot) = f_s(\cdot)1_0$, where 1_o is an all zero vectors with only the first element being 1.

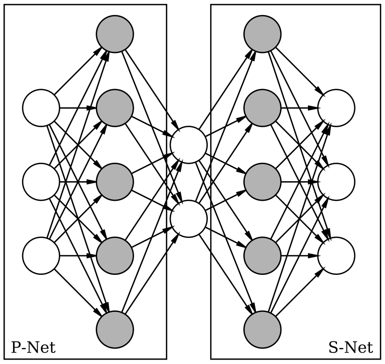


Figure 1: Nonlinear factorization, where grey units indicating nonlinear units while white ones are linear units.

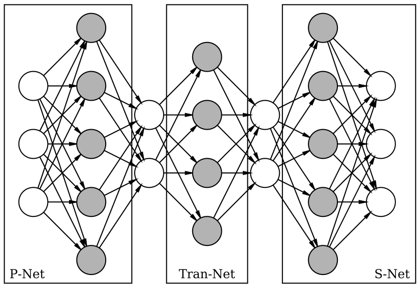
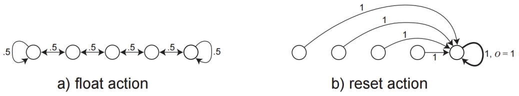


Figure 2: Network for the transition functions (grey layer indicates nonlinear units while white ones are linear).

3. Experiment

For the testing, we use the domain of float-reset problem, proposed in [Littman et al.]. An interesting fact about this problem is that, it is known for having a smaller model size(number of core tests required) when the prediction is nonlinear instead of linear [Littman et al.]. Therefore, we choose this domain as our test setting to see if our model can outperform linear PSR with smaller size.

To briefly introduce the float reset problem, as shown Figure 3, let us consider the float/reset problem consisting of a linear string of 5 states with a distinguished reset state on the far right. One action, f (float), causes the system to move uniformly at random to the right or left by one state, bounded at the two ends. The other action, r (reset), causes a jump to the reset state irrespective of the current state. The observation is always 0 unless the r action is taken when



a) float action

b) reset action

Figure 3: Float-reset problem.

the system is already in the reset state, in which case the observation is 1.

The training samples consist of 20,000 examples generated by this process with 200 states, which have random length. The testing samples have size of 1,000, in which 500 are actually from the float-reset process while the other 500 are randomly generated. The task is to classify these two different types of data.

For the neural network setting, the number of hidden units of P-Net and S-Net is set to $2,000$, while the number of hidden units in the Tran-Net is set to $2k$. Optimization method of training the neural net is AdamMax with learning rate of 0.01. We conducted experiments with three different model configuration: only applying non-linearity in regression step, which we refer to as non-reg; only applying non-linearity in factorization step, which we refer to as non-fac; and finally applying non-linearity in both of these two phases, and we call it non-both.

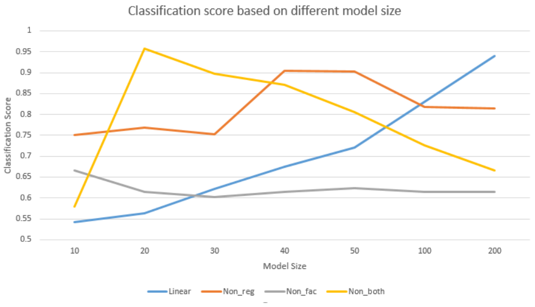


Figure 4: The classification score for float-reset problem based on different model size, ranging from 10 to 200.

As shown in Figure 4, the linear PSR has relatively low performance when the model size is small, and be able to reach near 0.9 accuracy when the model size reaches to 200, which is the number of states of the original float-reset process. However, for nonlinear PSR, it turns out to perform generally better than the linear one when the model size is small. For non-reg, we can see when the model size reaches to 40 to 50, it reaches its best performance of near 0.9 and then it declines as the model size increasing. For non-both, surprisingly, it reaches 0.95 with number of states being 20. However, for non-fac, it seems it cannot capture the system dynamics and fails to have any valid prediction, with around 0.5 score being at the same level of random guess. Therefore, for the float-reset problem, applying non-linearity in the regression step can definitely reduce the model size, while for the non-linearity in factorization, i.e. finding the nonlinear combination of sufficient statistics, it can help reduce the model size significantly if we also use nonlinear regression, but it also fails to capture the relationship. One remark to make is that, as we mentioned previously, f_p incorporates the nonlinear embedding of the sufficient statistics, while f_s is actually the stacked prediction functions. By only introducing non-linearity in factorization, while assuming the prediction functions are linear might be contradictory. One possible solution could be when only applying non-linearity in factorization, we set f_s to be just linear function.

4. Unfinished business(with Q learning)

This part I couldn't make it work, probably there is a bug in my code, but I ran out of time.

A natural extension of PSR could be use it as a state representation into the Q learning framework(with function approximator). In the experiment, I wanted to use a linear Q function with linear and nonlinear state representation of different model size to solve the task of float-reset problem, where now the reward is 1 if the observation is one. Clearly, the best policy would be, starting at whatever state, applying reset and the reset again and again to repeatedly gain reward. This policy, however, are learned through both linear and nonlinear state representation with any given model size. This is nevertheless rather impossible, as we have seen the poor performance shown previously for the linear representation when the model size is restricted. However, I am also not very sure if the policy is too easy to learn, as one can just simply always apply reset. I will try to solve the problem afterwards, as I really want to see what will happend with this nonlinear PSR with planning.

5. Conclusion and future work

In this report, we have introduced a nonlinear predictive state representation with a neural network based learning algorithm. Our learning algorithm can be seen as applying non-linearity into the factorization and regression steps of the spectral learning algorithm. Empirical results showed that our model can perform better for complicated intrinsic structures and can cope with restricted number of states.

In future works, we intend to further assess the benefits of introducing non-linearity in PSR on real word data and modify it properly to incorporate with planning.

References

Michael R James and Satinder Singh. Learning and discovery of predictive state representations in dynamical systems with reset. In *Proceedings of the twenty-first international conference on Machine learning*, page 53. ACM, 2004.

Michael L Littman, Richard S Sutton, Satinder Singh, et al. Predictive representations of state.

Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *Proceedings of the twenty-first international conference on Machine learning*, page 88. ACM, 2004.