Learning Multi-Step Predictive State Representations

Anonymous Submission

Abstract

Recent years have seen the development of efficient and provably correct spectral algorithms for learning models of partially observable environments arising in many applications. But despite the high hopes raised by this new class of algorithms, their practical impact is still below expectations. One reason for this is the difficulty in adapting spectral methods to exploit structural constraints about different target environments which can be known beforehand. A natural structure intrinsic to many dynamical systems is a multi-resolution behaviour where interesting phenomena occur at different time scales during the evolution of the system. In this paper we introduce the multi-step predictive state representation (M-PSR) and an associated learning algorithm that finds and leverages frequent patterns of observations at multiple scales in dynamical systems with discrete observations. We perform experiments on robot exploration tasks in a wide variety of environments and conclude that the use of M-PSR improves over the classical PSR for varying amounts of data, environment sizes, and number of observations symbols.

Introduction

Learning models of partially observable dynamical systems arises in many practical applications, including robotics, medical monitoring, market analysis etc. An approach which has interesting theoretical properties is that of predictive state representations (PSRs) (Littman, Sutton, and Singh 2001; Singh, James, and Rudary 2004; Rosencrantz, Gordon, and Thrun 2004), in which the model consists off a set of predictions about the future evolution of the system, conditioned on past outcomes. We focus on linear PSRs, in which predictions for any future trajectories can be obtained as a linear combination of a finite set of "core" predictions. In recent work, a variety of spectral learning algorithms have been proposed for tackling this problem, e.g. (Boots, Siddiqi, and Gordon 2011; Hamilton, Fard, and Pineau 2013). Such algorithms are appealing in part because of their strong theoretical properties, such as statistical consistency and learning efficiency (Hsu, Kakade, and Zhang 2009; Bailly, Habrard, and Denis 2010). Unfortunately, the uptake of these algorithms in practice is still limited. This is due partly to the fact that existing spectral learning algorithms are general-purpose, and do not explicitly try to leverage regularities in the data. However, applications often have certain types of structure which, if taken into account, can facilitate more efficient learning.

In this paper, we focus on a specific type of structure which arises frequently in dynamical systems: behaviour which takes place at different time scales. In signal processing, this is often the type of structure leveraged to provide efficient algorithms. Similar efficiencies can be obtained by using multiple time scales in planning and reinforcement learning (Sutton, Precup, and Singh 1999). We propose a new model of PSRs for environments with discrete observations, which we call the multi-step PSR (M-PSR). Intuitively, an M-PSR includes not only single-step observations, but also contiguous sequences of observations. This allows capturing structure occurring over a larger time scale. We show how the standard spectral learning for PSRs extends to M-PSRs, and present a data-driven algorithm for selecting a particular M-PSR based on sampled data. Empirical results in simulated navigation tasks show that M-PSRs improve the quality and speed of learning, compared to classical PSRs, over a range of environment sizes, number of observation symbols and amounts of data.

The Multi-Step PSR

A linear *predictive state representation* (PSR) for an autonomous dynamical system with discrete observations is a tuple $\mathcal{A} = \langle \Sigma, \boldsymbol{\alpha}_\epsilon, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma\in\Sigma}\rangle$ where: Σ is a finite set of possible observations, $\boldsymbol{\alpha}_\epsilon, \boldsymbol{\alpha}_\infty \in \mathbb{R}^n$ are vectors of initial and final weights, and $\mathbf{A}_\sigma \in \mathbb{R}^{n\times n}$ are the transition operators associated with each possible observation. The dimension n is the number of states of $\mathcal{A}$. Formally, a PSR is a *weighted finite automata* (WFA) (Droste and Vogler 2009) computing a function given by the probability distribution of sequences of observations in a partially observable dynamical system with a finite number of states. The function $f_\mathcal{A} : \Sigma^\star \to \mathbb{R}$ computed by $\mathcal{A}$ is given by:

$$f_\mathcal{A}(x) = f_\mathcal{A}(x_1 \cdots x_t) = \boldsymbol{\alpha}_\epsilon^\top \mathbf{A}_{x_1} \cdots \mathbf{A}_{x_t} \boldsymbol{\alpha}_\infty = \boldsymbol{\alpha}_\epsilon^\top \mathbf{A}_x \boldsymbol{\alpha}_\infty \ .$$

The value of $f_\mathcal{A}(x)$ is interpreted as the probability that the system produces the sequence of observations $x = x_1 \cdots x_t$ starting from the initial state specified by $\boldsymbol{\alpha}_\epsilon$. Depending on

the model, this can be a probability that the system generates x and *stops*, or the probability that the system generates x and *continues*. Given a partial history u of the evolution of the system – i.e. a prefix in string terminology – the state of a PSR can be updated from $\boldsymbol{\alpha}_\epsilon$ to $\boldsymbol{\alpha}_u = \boldsymbol{\alpha}_\epsilon \mathbf{A}_u$. This allows for conditional queries about the possible observations that will follow u. For example, the probability of observing the string v given that we have already observed u will be written as:

$$f_{\mathcal{A},u}(v) = \frac{\boldsymbol{\alpha}_u \mathbf{A}_v \boldsymbol{\alpha}_\infty}{\nu_{\mathcal{A}}(u)} \ ,$$

where $\nu_{\mathcal{A}}(u)$ is a normalizing constant to obtain a proper conditional distribution whose expression depends on the actual semantics of the PSR (stop vs. continuation probabilities).

To define a multi-step PSR, we augment a PSR with two extra objects: a set of *multi-step observations* $\Sigma' \subset \Sigma^+$ containing non-empty strings formed by basic observations, and a *coding function* $\kappa : \Sigma^\star \to \Sigma'^\star$ that, given a string of basic observations, produces an equivalent string composed using multi-step observations. The choice of Σ' and κ will typically be customized for a given application, in order to reflect the particular patterns expected to arise in different environments. However, we assume these objects satisfy a basic set of requirements, for the sake of simplicity and to avoid degenerate situations:

1. The set Σ' must contain all symbols in Σ; i.e. $\Sigma \subseteq \Sigma'$
2. The function κ satisfies $\partial(\kappa(x)) = x$ for all $x \in \Sigma^\star$, where $\partial : \Sigma'^\star \to \Sigma^\star$ is the *decoding morphism* between free monoids given by $\partial(z) = z \in \Sigma^\star$ for all $z \in \Sigma'$. Note this implies that $\kappa(\epsilon) = \epsilon$, $\kappa(\sigma) = \sigma$ for all $\sigma \in \Sigma$, and κ is injective.

Using these definitions, a *multi-step PSR* (M-PSR) is a tuple $\mathcal{A}' = \langle \Sigma, \Sigma', \kappa, \boldsymbol{\alpha}_\epsilon, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma \in \Sigma'} \rangle$ containing a PSR with observations in Σ', together with the basic observations Σ and the corresponding coding function κ. In addition to the function $f_{\mathcal{A}'} : \Sigma'^\star \to \mathbb{R}$ which $\mathcal{A}'$ computed by virtue of being a PSR over Σ', it also computes another function $f'_{\mathcal{A}'} : \Sigma^\star \to \mathbb{R}$ given by $f'_{\mathcal{A}'}(x) = f_{\mathcal{A}'}(\kappa(x))$. In many cases we will abuse notation and write $f_{\mathcal{A}'}$ for $f'_{\mathcal{A}'}$ when there is no risk of confusion.

Examples of M-PSRs

We now describe several examples of M-PSR, putting special emphasis on models that will be used in our experiments.

Base M-PSR A PSR with a single observation $\Sigma = \{\sigma\}$ can be used to measure the time – i.e. number of discrete time-steps – until a certain event happens (Bacon, Balle, and Precup 2015). In this case, a natural approach to build an M-PSR for timing models is to build a set of multi-step observations containing sequences of a's whose lengths are powers of a fixed base. That is, given an integer $b > 0$, we build the set of multi-step observations as $\Sigma' = \{\sigma, \sigma^b, \sigma^{b^2}, \ldots, \sigma^{b^K}\}$ for some positive K. A natural choice of coding map in this case is the one that represents any length $t \geq 0$ as a number in base b, with the difference that the largest power b that is allowed is b^K. This corresponds to writing (in a unique way) $t = t_0 b^0 + t_1 b^1 + t_2 b^2 + \cdots + t_K b^K$, where $0 \leq t_k \leq b-1$ for $0 \leq k \leq K$, and $t_K \geq 0$. With this decomposition we obtain the coding map $\kappa(\sigma^t) = (\sigma^{b^K})^{t_K} (\sigma^{b^{K-1}})^{t_{K-1}} \cdots (\sigma^b)^{t_1} (\sigma)^{t_0}$. Note that we choose to write powers of longer multi-step observations first, followed by powers of shorter multi-step observations. For further reference, we will call this model the Base M-PSR.

For the multiple observation case, one can also use a Base M-PSR. For example, if there are two observations $\Sigma = \{\sigma_1, \sigma_2\}$, we can take $\Sigma' = \{\sigma_1, \sigma_2, \sigma_1^b, \sigma_2^b, \sigma_1^{b^2}, \sigma_2^{b^2}, ..., \sigma_1^{b^K}, \sigma_2^{b^K}\}$. This can of course be extended for any finite number of observations. For the encoding map κ we first split the string into sequences of consecutive repeated symbols and then use the same encoding as for timing. For example: $\kappa(\sigma_1^5 \sigma_2^3) = (\sigma_1^2)^2 (\sigma_1)(\sigma_2^2)(\sigma_2)$ when using $b = 2$ and $K = 1$.

Tree M-PSR Another example for the multiple observation case is what we call the Tree M-PSR. In a Tree M-PSR, we set $\Sigma' = \{x \in \Sigma^\star, |x| \leq L\}$, where L is a parameter. Note however that $|\Sigma'| = O(|\Sigma|^L)$, so in practice L must remain small if we want the M-PSR to be representable using a small number of parameters. For the decoding map κ, we first split a string x as $x = u_1 u_2 \cdots u_m u_f$, where $|u_i| = L$ for $1 \leq i \leq n$ and $|u_f| = |x| - (m \cdot L) < L$, then we set $\kappa(x) = (u_1)(u_2) \cdots (u_m)(u_f)$. Note that this encoding promotes the use of multi-step symbols corresponding to longer strings more often than those symbols corresponding to shorter strings.

Data-Driven M-PSR Although the constructions above have some parameters that can be tuned depending on the particular application, they are not directly dependent on the observations produced by the target environment. In our experiments, the performance of M-PSRs depends heavily on how Σ' reflects the observations. Thus, we develop an algorithm for choosing Σ' in a data-driven way. In addition, we provide a generic coding function κ that can be applied to any M-PSR. This encoding delivers good experimental predictive performance and computationally inexpensive probability queries. Together, these yield another type of M-PSR, which we call the Data-Driven M-PSR, which is the output of the learning algorithm described in the next section.

Learning Algorithm for M-PSR

In this section, we describe a learning algorithm for M-PSR which combines the standard spectral method for PSR (Boots, Siddiqi, and Gordon 2011) with a greedy algorithm for extracting an extended set of symbols Σ' containing frequent patterns of observations and which minimizes a coding cost for a generic choice of coding function κ.

Spectral Learning Algorithm

We start by extending the spectral learning algorithm to M-PSR under the assumption that κ and Σ' are known. In this case, the learning procedure only needs to recover the operators $\mathbf{A}_\sigma$ for all $\sigma \in \Sigma'$, and the initial and final weights $\boldsymbol{\alpha}_\epsilon, \boldsymbol{\alpha}_\infty$. We start by recalling some basic notation about

Hankel matrices (Carlyle and Paz 1971; Fliess 1974), and then proceed to describe the learning algorithm. For the purpose of describing the learning algorithm, we start by assuming that the function $f : \Sigma^\star \to \mathbb{R}$ associated with the target M-PSR can be evaluated for every string. This does not necessarily imply that we know the M-PSR, since the values of f correspond to probabilities of observations that can be effectively estimated from data.

Given $f : \Sigma^\star \to \mathbb{R}$, we will use its *Hankel matrix* representation $\mathbf{H}_f \in \mathbb{R}^{\Sigma^\star \times \Sigma^\star}$, which is an infinite matrix whose rows and columns are indexed by strings in $\Sigma^\star$ and whose entries are given by $\mathbf{H}_f(u, v) = f(uv)$. To efficiently work with this matrix, we only consider finite sub-blocks indexed by a finite set of prefixes $\mathcal{P} \subset \Sigma^\star$ and suffixes $\mathcal{S} \subset \Sigma^\star$. Both $\mathcal{P}$ and $\mathcal{S}$ are input parameters given to the algorithm; see (Balle, Quattoni, and Carreras 2012) for a discussion on how to choose these in practice. The pair $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is sometimes called a basis, and it determines a sub-block $\mathbf{H}_\mathcal{B} \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ of $\mathbf{H}_f$ with entries given by $\mathbf{H}_\mathcal{B}(u, v) = \mathbf{H}_f(u, v)$ for all $u \in \mathcal{P}$ and $v \in \mathcal{S}$. For a fixed basis, we also consider the vectors $\mathbf{h}_\mathcal{S} \in \mathbb{R}^\mathcal{S}$ with entries given by $\mathbf{h}_\mathcal{S}(v) = \mathbf{H}_f(\epsilon, v)$ for every $v \in \mathcal{S}$, and $\mathbf{h}_\mathcal{P} \in \mathbb{R}^\mathcal{P}$ with $\mathbf{h}_\mathcal{P}(u) = \mathbf{H}_f(u, \epsilon)$.

Note that the definitions above have only used Σ. In order to recover operators $\mathbf{A}_\sigma$ for all $\sigma \in \Sigma'$ we will need to consider multi-step shifts of the finite Hankel matrix $\mathbf{H}_\mathcal{B}$. In particular, given $\sigma \in \Sigma'$ we define the sub-block $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ whose entries are given by $\mathbf{H}_\sigma(u, v) = f(u\, \sigma\, v)$. Note that this can be interpreted as either using the lift $f(\kappa(u)\, \sigma\, \kappa(v))$ or the decoding $f(u\, \partial(\sigma)\, v)$, but the actual value in the matrix $\mathbf{H}_\sigma$ will be the same.

Now we can give the details of the learning algorithm. Suppose the basis $\mathcal{B}$ and the desired number of states n are given. We start by collecting a set of sampled trajectories and use them to estimate the matrices $\mathbf{H}_\mathcal{B}, \mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ and vectors $\mathbf{h}_\mathcal{P} \in \mathbb{R}^\mathcal{P}$, $\mathbf{h}_\mathcal{S} \in \mathbb{R}^\mathcal{S}$. Then, we take the truncated SVD $\mathbf{U}_n \mathbf{D}_n \mathbf{V}_n^\top$ of $\mathbf{H}_\mathcal{B}$, where $\mathbf{D}_n \in \mathbb{R}^{n \times n}$ contains the first n singular values of $\mathbf{H}_\mathcal{B}$, and $\mathbf{U}_n \in \mathbb{R}^{\mathcal{P} \times n}$ and $\mathbf{V}_n \in \mathbb{R}^{\mathcal{S} \times n}$ contain the first left and right singular vectors respectively. Finally, we compute the transition operators of the M-PSR as $\mathbf{A}_\sigma = \mathbf{D}_n^{-1} \mathbf{U}_n^\top \mathbf{H}_\sigma \mathbf{V}_n$ for all $\sigma \in \Sigma'$, and the initial and final weights as $\boldsymbol{\alpha}_\epsilon^\top = \mathbf{h}_\mathcal{S}^\top \mathbf{V}_n$ and $\boldsymbol{\alpha}_\infty = \mathbf{D}_n^{-1} \mathbf{U}_n^\top \mathbf{h}_\mathcal{P}$. This algorithm yields an M-PSR with n states. It was proved in (Boots, Siddiqi, and Gordon 2011) that this algorithm is statistically consistent for PSRs (under a mild condition on $\mathcal{B}$) and the same guarantees extend to M-PSR.

A Generic Coding Function

Given Σ and Σ', a generic coding function $\kappa : \Sigma^\star \to \Sigma'^\star$ can be obtained by minimizing the coding length $|\kappa(x)|$ of every string $x \in \Sigma^\star$. More formally, we consider the coding κ given by

$$\kappa(x) = \operatorname*{argmin}_{y \in \Sigma'^\star,\ \partial(y) = x} |y| \ .$$

Note that for the single observation case, this is equivalent to the optimal coin change problem, which is a textbook example of dynamic programming. This has advantage of minimizing the number of operators $\mathbf{A}_\sigma$ that will need to be multiplied to compute the value of the M-PSR on a string x. At the same time, operators expressing long transition sequences capture intermediate contributions of all states even if chooses to use a small model. Intuitively, this should provide M-PSRs with better performance for smaller models.

The optimization problem above can be solved by dynamic programming. To do so, one inductively computes the optimal string encoding for the prefix $x_1 \cdots x_i$ for all $1 \leq i \leq |x|$. This can be obtained by minimizing over all $\sigma \in \Sigma'$ which terminate at the index i of x. We provide the full pseudo-code for this encoding function in the first section of the appendix.

Greedy Selection of Multi-Step Observations

The multi-step transition sequences that need to be added to Σ' can be selected in a data-driven fashion using a greedy algorithm. Having a Σ' which reflects the type of observations produced the target system will promote short encodings when coupled with the coding function described above. In practice, this greedy algorithm picks substrings appearing in the training data which are long, frequent, and diverse. From an intuitive standpoint, one can view structure in observation sequences as relating to the level of entropy in the distribution over multi-step observations produced by the system.

We now provide a general description of how the algorithm works. Detailed pseudo-code is presented in the supplementary material. The algorithm starts by finding all possible substrings in $\Sigma^\star$ that appear in the training dataset. As a preprocessing step, and for computational reasons, this is constrained to contain only the the M most frequent substrings, where M is a parameter chosen by the user. Here the frequency is measured by number of observed trajectories that contain a given substring.

The construction of Σ' is initialised by Σ and a new multi-step symbol is added at each phase. A phase starts by evaluating each substrings in terms of how much reduction it would cause in the number of transition operators used to encode the whole training set if the symbol were to be added to Σ'. The algorithm then adds to Σ' a multi-step symbol corresponding to the best substring, i.e. the one that would reduce the most the whole coding cost (obtained by applying κ). More formally, at the ith iteration the algorithm finds:

$$\operatorname*{argmin}_{u \in \mathrm{sub}_M} \sum_{x \in \mathrm{train}} |\kappa_{\Sigma'_i \cup \{u\}}(x)| \ ,$$

where train is the training set, sub_M is the set of substrings under consideration of length at most M, Σ'_i is the set of multi-step observations at the beginning of phase i and we use $\kappa_{\Sigma'}$ to denote the encoding function with respect to a given set of multi-step observations for clarity. The algorithm terminates after Σ' reaches a predetermined size.

Experiments

In this section, we assess the performance of PSRs and different kinds of M-PSRs in different types of environments, varying several parameters including the model sizes and number of observations used. For all the plots, the x-axis is model size of the PSR/M-PSRs and the y-axis is an error measurement of the learned PSR/M-PSRs.

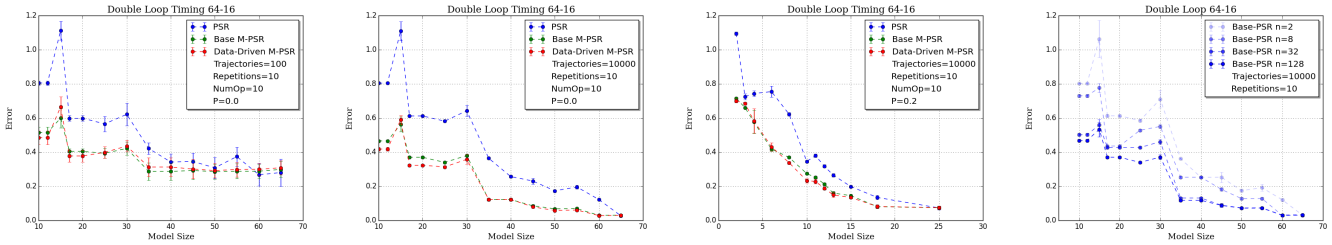

Figure 1: Results for the Double Loop of size 64-16 with (in order) a low amount of data, high amount of data and varying amount of noise (right two panels)

In all the experiments, an agent is positioned in a partially observable environment and navigates stochastically based on a transition function $\delta : S \times S \rightarrow [0, 1]$. An observation symbol is produced on every transition. When the agent exits, we record the concatenation of the symbols produced, which represents the observation sequence for the completed trajectory. We perform experiments in two environments: a Double Loop maze and a Pacman-style environment. These are both depicted in the supplementary material.

For the Base M-PSR (i.e., the timing models), we construct the empirical Hankel matrix by taking $\mathcal{P}, \mathcal{S} = \{a^i, \forall i \leq n\}$, where n is an application-dependent parameter. For Double Loop environments we set $n = 150$, while for the Pacman domain, $n = 600$. For these choices of n, we verify that as the amount of data increases, the learned PSR with the true model size converges to the true model. For the Base M-PSR, we set $b = 2, K = 8$, so that the longest string in Σ' is a^{256}.

For the tasks with multiple observations, a slightly more complex approach is required to choose $\mathcal{P}$ and $\mathcal{S}$. For the prefixes $\mathcal{P}$, we select the k most frequent prefixes from our observation set. For the suffixes $\mathcal{S}$, we take all suffixes that occur in $\mathcal{P}$. We also require prefix completeness. That is, if p' is a prefix of $p \in \mathcal{P}$, then $p' \in \mathcal{P}$. This heuristic for constructing empirical Hankel matrices was given in previous work (Balle, Quattoni, and Carreras 2012). For the Base M-PSR, we take $K = 8$ and $B = 2$ for both symbols $\Sigma = \{g, b\}$, where g stands for green and b for blue. For the Tree M-PSR, we set $L = 7$ for a total of 128 operators, a far larger limit than for the other M-PSRs.

To measure the performance of a PSR/M-PSR we use the following norm:

$$\|f - \hat{f}\|_2 = \sqrt{\sum_{x \in \Sigma^\star} (f(x) - \hat{f}(x))^2} \ ,$$

where f denotes the true probability distribution over observations and $\hat{f}$ denotes the function associated with the learned M-PSR/PSR. In our environments, f can be computed exactly, as we have access to the underlying HMMs.

Since the set of observations $\Sigma^\star$ is infinite, we compute approximations to this error norm by fixing a set of strings T and summing over it. For the Base M-PSR (timing) case, we take $T = \{a^i, \forall i \leq C\}$ with $C = 600$. Importantly, C has to be large enough, such that $\sum_{i=0}^{C} f(a^i) > 0.99$. For the multiple observation case, we take all possible strings that can be produced from the prefixes and suffixes present in our dataset: $T = \{ps, \forall p \in \mathcal{P}, \forall s \in \mathcal{S}\}$

Double Loop Timing

We start by considering the Double Loop environment, and the task of learning timing models. The length of a loop is the number of states in the loop. A trajectory begins with the agent starting at the intersection of the two loops. Here, the agent has a 50% probability of entering either loop. At intermediate states, the agent moves to the next state in the loop with probability $1 - P$ or remains in its current state with probability P. Exit states are located halfway into each loop. The agent leaves the environment at exit states with 50% probability. This means that if loop lengths are $l_1 = 64$ and $l_2 = 16$, we observations will come in the form of $n_1 l_1 + n_2 l_2 + (1-\alpha) l_1/2 + \alpha l_2/2$, with $\alpha = 1$ representing an exit through loop 1 and $\alpha = 0$ an exit through loop 2.

Figure 1 provides results for the case in which one loop consists of 64 states and the other of 16 states. The leftmost panel presents the results obtained by learning from 100 observation sequences, while the results in the next panel are obtained using 10000 observations. In both cases, the M-PSR models outperform the simple PSR for small model sizes. As expected, with more data, the error drops close to 0 for the larger models. The right panels show the effect of varying the self-transition probability P, in order to to simulate noise in the environment. The two panels show curves of $P = 0.2$ and $P = 0$ respectively. We find that the environment with noisy loops is more compressible, that is, one can achieve better performance for low model sizes, but the performance becomes worse as the model size attains the environment's true size. M-PSRs still significantly outperform the standard PSR for reduced model sizes. The rightmost panel also illustrates the effect of K on the performance of the Base M-PSRs. We note that larger values of K have better performance, up to $K = 7$.

In Figure 2, we illustrate how varying the number of multi-step transition operators affects performance for the 64-16 double loop. As expected, a higher number of operators improves performance, but the effect tapers off at about 20 operators. Here the most important operators seem to be: $\{a, a^8, a^{24}, a^{32}, a^{72}\}$ which are closely tied to the environment's periodic structure.

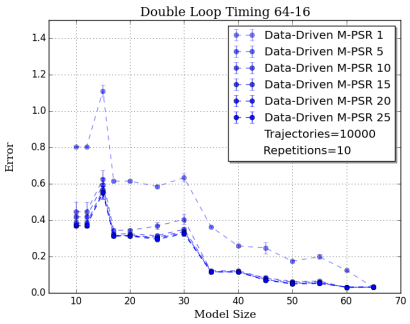

Figure 2: Varying NumOps

Loop Lengths

In Figure 3, we plot the results of a 47-27 labyrinth. Here, because of the lengths of the loops, the observations will not be as compactly expressed from the Base M-PSR. Again, M-PSRs outperform the standard PSR for reduced model sizes. Additionally, the Data-Driven M-PSR does better than the Base M-PSR.

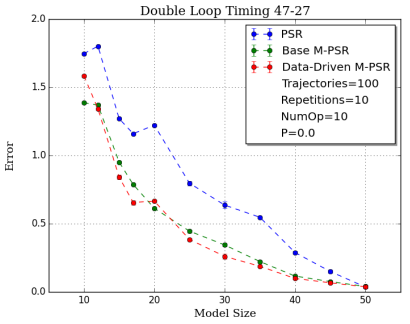

Figure 3: High Data Double Loop 47-27

Large Labyrinth Timing

We now turn our attention to a larger labyrinth environment, similar to a Pacman game (depicted in the supplementary material). Transitions to new states occur with equal probability. The weight $w(u, v)$ between states u and v corresponds to the number of time steps taken to get from u to v. We add an additional parameter sF, called the stretch factor, which is used to scale all of the weights in the graph.

In Figures 4 and 5 we vary the number of observations used for learning. M-PSRs outperform the traditional PSR regardless of the amount of data. Secondly, as expected, the performance of all models is better with more training data.

In Figures 5,6 and 7 we vary the stretch factor parameter, while keeping the size of the dataset fixed. We find that a higher values of sF (such as in 5 provide increased improvement of the M-PSR relative to the performance of the standard PSR.

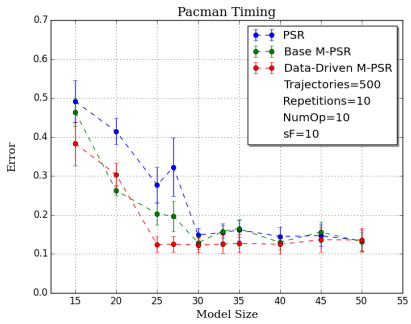

Figure 4: Low Data Pacman Labyrinth

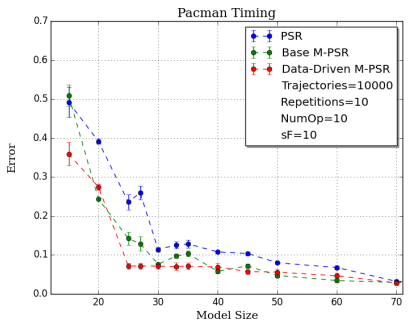

Figure 5: High Data Pacman Labyrinth

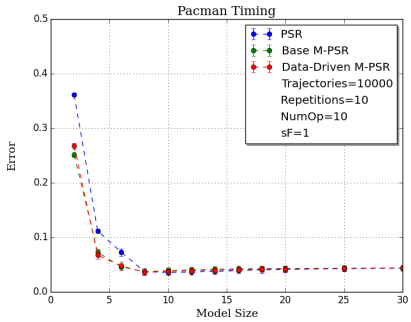

Figure 6: Stretch Factor: 1

Multiple Observations: Coloured Loops

We now move to the multiple observation case. We construct a Double Loop environment where observations in the first loop are all is green, with the length of the loop being $l_1 = 27$, and the second loop is blue, with length $l_2 = 17$. We fix the length of each trajectory at $3(l_1 + l_2)$

We build empirical estimates for the Hankel matrix as follows:

$$f(x) = \frac{|\text{train} \cap x\Sigma^\star|}{|\text{train} \cap \Sigma^{\geq |x|}|} \ .$$

This means that the PSRs will compute the probability of x occurring as a prefix.

As for the timing case, we vary the amount of data used to

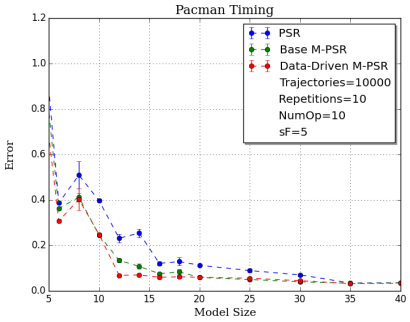

Figure 7: Stretch Factor: 5

learn the PSRS/M-PSRs in Figures 8 and 9. Once again, we find M-PSRs perform far better, especially the Data-Driven M-PSR. This makes sense, because when the complexity in the observations increases, only custom M-PSRs will express transitions compactly.

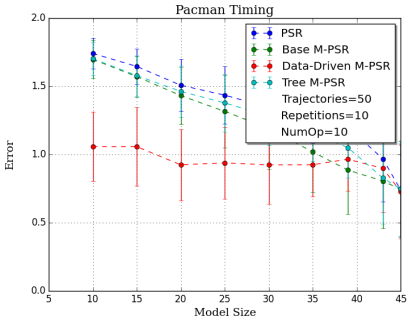

Figure 8: Low Data Colored Loops 27-17

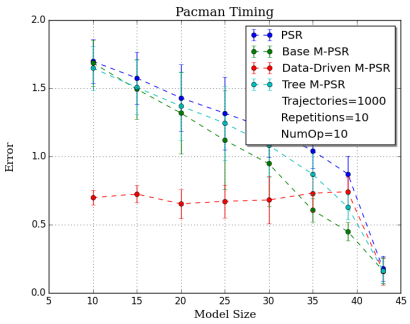

Figure 9: High Data Colored Loops 27-17

In Figure 10, we vary the number of multi-step transition operators learned. The important operators emerging from the learning process are $\{g, b, g^{27}, b^{17}\}$, which is again very encouraging, as it reflects the structure of this particular environment.

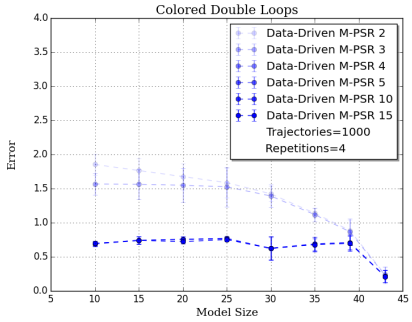

Figure 10: Varying NumOps

Discussion

We presented an approach to leveraging multiple temporal scales in the behaviour of dynamical systems, in order to learn predictive state representations. The proposed model, M-PSR, can be learned by adapting existing spectral approaches. To our knowledge, the only other work that attempts to include temporal abstraction in PSRs is due to (Wolfe and Singh 2006), who's use temporally extended actions, or options (Sutton, Precup, and Singh 1999), and build PSRs on top of these. However, this model has a very different flavour from our approach, as we bundle together observations into multiple steps. In particular, our approach is applicable even when there are no actions, such as in the case of HMMs, whereas this previous work requires structure in the action space. In all the experiments we conducted, M-PSRs offer significantly better predictive performance for reduced model sizes than PSRs. In addition, Data-Driven PSRs offer improvements over generic M-PSRs by learning the transition operators specific to the environment, which is very important when prior information about appropriate time scales for modelling is not known. Our evaluation focused on illustrating the advantage of the proposed model especially when the amount of data available is small, and in noisy environments. Although this was not specifically illustrated in the experiments. M-PSRs also offer a computational advantage when performing conditional probability queries, as they use far fewer matrices than regular PSRs.. This can be very important for applications in which queries are done online, such as in planning environments.

We presented a comprehensive set of experiments but in simulated domains, in order to have some ground truth available. However, we anticipate that the proposed models would be useful in real tasks, for example in financial market prediction. In this case, one could discretize market levels into bins, and use M-PSRs to learn predictive models over multiple time steps, in order to capture both long-term trends and fast fluctuations. We also plan to experiment with this approach in the domain of processing physiological signal recordings, which similarly exhibit short and long-term trends. On the theoretical side, it would be interesting to analyze the existence of an "optimal" set of symbols for the data-driven M-PSR, and to develop further algorithms for finding it.

References

Bacon, P.-L.; Balle, B.; and Precup, D. 2015. Learning and planning with timing information in Markov Decision Processes. In *UAI*.

Bailly, R.; Habrard, A.; and Denis, F. 2010. A spectral approach for probabilistic grammatical inference on trees. In *ALT*.

Balle, B.; Quattoni, A.; and Carreras, X. 2012. Local loss optimization in operator models: A new insight into spectral learning. *International Conference on Machine Learning (ICML)*.

Boots, B.; Siddiqi, S.; and Gordon, G. 2011. Closing the learning planning loop with predictive state representations. *International Journal of Robotic Research*.

Carlyle, J. W., and Paz, A. 1971. Realizations by stochastic finite automata. *Journal of Computer Systems Science*.

Droste, M. Kuich, W., and Vogler, H. 2009. *Handbook of weighted automata*. Springer.

Fliess, M. 1974. Matrices de Hankel. *Journal de Mathématiques Pures et Appliquées*.

Hamilton, W. L.; Fard, M. M.; and Pineau, J. 2013. Modelling sparse dynamical systems with compressed predictive state representations. In *ICML*.

Hsu, D.; Kakade, S.; and Zhang, T. 2009. A spectral algorithm for learning Hidden Markov Models. In *COLT*.

Littman, M. L.; Sutton, R. S.; and Singh, S. 2001. Predictive representations of state. *Neural Information Processing Systems Conference (NIPS)*.

Rosencrantz, M.; Gordon, G.; and Thrun, S. 2004. Learning low dimensional predictive representations. In *ICML*.

Singh, S.; James, M. R.; and Rudary, M. R. 2004. Predictive state representations: A new theory for modeling dynamical systems. In *UAI*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*.

Wolfe, B., and Singh, S. 2006. Predictive state representations with options. *International Conference on Machine Learning (ICML)*.