Learning Multi-Step Predictive State Representations

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-PSRs improves over the classical PSR for varying amounts of data, environment sizes, and number of observations symbols.

1 Introduction

Learning models of partially observable dynamical systems is a problem that arises in many practical applications, including robotics, medical monitoring, market analysis etc. Predictive state representations (PSRs) [Littman *et al.*, 2001; Singh *et al.*, 2004; Rosencrantz *et al.*, 2004], provide a theoretically justified approach in which at any time-step the current state is modeled as a set of predictions about the future evolution of the system, conditioned on past outcomes. The most popular of these models are linear PSRs, in which predictions for any future trajectories can be obtained as a linear combination of a finite set of "core" predictions. A variety of spectral algorithms for learning linear PSRs have been proposed in recent works, e.g. [Boots *et al.*, 2011; Hamilton *et al.*, 2013].

Spectral algorithms are appealing because of their strong theoretical properties, such as statistical consistency and learning efficiency [Hsu *et al.*, 2009; Bailly *et al.*, 2010]. Unfortunately, their practical uptake is still limited, in part, due to the fact that these general-purpose algorithms are not designed to leverage the structural regularities frequently found in applications. This is a challenging question that needs to be tackled in order to facilitate more efficient spectral learning algorithms for specific applications. Recent work along these lines has focused on special structures encountered in epigenomics [Zhang *et al.*, 2015]. 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. This type of structure is often leveraged to provide efficient algorithms in signal processing. Similar gains can be obtained by using multiple time scales in planning and reinforcement learning, e.g. [Sutton *et al.*, 1999].

Our approach is based on a new class of PSRs which we call the multi-step PSR (M-PSR). Like the classical linear PSR, M-PSRs represent the dynamics of a partially observable system with discrete observations. However, unlike PSRs, M-PSRs are able to capture structure occurring at multiple time scales by representing transitions between states over multi-step observations. Our main result is a spectral learning algorithm for M-PSRs combined with a data-driven strategy for multi-step observation selection. Through an extensive empirical evaluation, we show that in environments where characteristic multi-step observations occur frequently, M-PSRs improve the quality of learning with respect to classical PSRs. This improvement is uniform over a range of environment sizes, number of observation symbols, and amounts of training data.

2 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 $f_\mathcal{A} : \Sigma^\star \to \mathbb{R}$ com-

puted 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 semantics of 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 the state of a PSR can be updated from $\boldsymbol{\alpha}_\epsilon$ to $\boldsymbol{\alpha}_u = \boldsymbol{\alpha}_\epsilon \mathbf{A}_u$. This update allows for conditional queries about future observations. For example, the probability of observing a string v given that we have already observed u is:

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

where $\nu_{\mathcal{A}}(u)$ is a normalizing constant.

To 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 takes a string of basic observations and produces an equivalent string composed of multi-step observations. The choice of Σ' and κ can be customized for each application, and will typically capture frequent patterns of observations arising in a particular environment. For the sake of simplicity and to avoid degenerate situations, we assume these objects satisfy the following requirements:

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 this notation, we define a *multi-step PSR* (M-PSR) as 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 standard PSR function $f_{\mathcal{A}'} : \Sigma'^\star \to \mathbb{R}$, an M-PSR also defines the 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.

2.1 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 *et al.*, 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 $b > 0$, the set of multi-step observations is $\Sigma' = \{\sigma, \sigma^b, \sigma^{b^2}, \ldots, \sigma^{b^K}\}$ 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, the coding map is given by $\kappa(\sigma^t) = (\sigma^{b^K})^{t_K} (\sigma^{b^{K-1}})^{t_{K-1}} \cdots (\sigma^b)^{t_1} (\sigma)^{t_0}$. 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*.

Base M-PSRs can also be extended to the case with multiple basic observations, $|\Sigma| > 1$. For example, if there are two observations $\Sigma = \{\sigma_1, \sigma_2\}$, we can take $\Sigma' = \{\sigma_1, \sigma_2, \sigma_1^b, \sigma_2^b, \sigma_1^{b^2}, \sigma_2^{b^2}, ..., \sigma_1^{b^K}, \sigma_2^{b^K}\}$. The encoding map κ first splits the string into sequences of consecutive repeated symbols and then uses the same encoding as before. 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

A more flexible generalization of the Base M-PSR 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. The decoding map κ first splits 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 $|u_f| = |x| - (m \cdot L) < L$, and then sets $\kappa(x) = (u_1)(u_2) \cdots (u_m)(u_f)$. Note that this encoding promotes the use of multi-step symbols corresponding to longer strings more often than those corresponding to shorter strings.

Data-Driven M-PSR

The constructions above have some parameters that can be tuned depending on the particular application, but in general they are not directly dependent on the sequences of observations more frequently produced by the target environment. Intuition suggests that the performance of M-PSRs will depend heavily on how well Σ' reflects frequent observation patterns; this is corroborated by the experiments in Section 4. Thus, we develop an algorithm for adaptively choosing Σ' from data. For this to work, we need to provide a generic coding function κ that can be applied to any M-PSR. This encoding needs to deliver good predictive performance and computationally cheap queries. We call the output of this learning process a *Data-Driven M-PSR*. The details of this approach are described in the next section.

3 Learning Algorithm for M-PSR

This section describes a learning algorithm for M-PSR combining standard spectral techniques [Boots *et al.*, 2011] with a algorithm for building an extended set of symbols Σ' containing frequent patterns of observations.

3.1 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 first recall some basic notation about Hankel matrices [Carlyle and Paz, 1971; Fliess, 1974], and then proceed to describe the learning algorithm. To simplify the description of the learning algorithm we assume that the function $f : \Sigma^\star \to \mathbb{R}$ associated with the target M-PSR can be evaluated for every string. In practice these are unknown, but can be effectively estimated from data because they correspond to probabilities of observations.

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 with rows and columns indexed by strings in $\Sigma^\star$ and whose entries satisfy $\mathbf{H}_f(u, v) = f(uv)$. 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 *et al.*, 2012] for a discussion on how to choose these in practice. The pair $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is 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 only depend on Σ. In order to recover operators $\mathbf{A}_\sigma$ for all $\sigma \in \Sigma'$ we 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 $\mathbf{H}_\sigma(u, v) = f(u\,\sigma\,v)$. 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.

Using the notation above we can give a simple description of the learning algorithm. Suppose the basis $\mathcal{B}$ and the desired number of states n are given. The algorithm starts by collecting a set of sampled trajectories and uses 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, it takes 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, it computes 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 *et al.*, 2011] that this algorithm is statistically consistent for PSRs (under a mild condition on $\mathcal{B}$). The same guarantees trivially extend to M-PSR.

3.2 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

$$\kappa(x) = \operatorname*{argmin}_{z \in \Sigma',\, x = yz,\, |y| < |x|} |\kappa(y)(z)|$$

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 can capture the contributions of all intermediate states even if the chosen model size is too small to represent every state traversed by a single-observation model. This is one of the reasons why M-PSRs show better performance than PSR for smaller models (see Section 4).

The pseudocode of the algorithm is given in Algorithm 1. It inductively computes the optimal string encoding for the prefix $x_1 \cdots x_i$ for all $1 \le i \le |x|$. This can be obtained by minimizing over all $\sigma \in \Sigma'$ which terminate at the index i of x. We use the following notation:

bestEncoding: A map from indices i of the query string x to the optimal encoding of $x[:i]$.

minEncoding: A map from indices i of the query string x to $|bestEncoding[i]|$

opsEnding: A map from indices i of x to the set of strings in Σ': $\{s \in \Sigma', x[i - |s| : i] = s\}$

Algorithm 1 Encoding Algorithm

INPUT: x
OUTPUT: $\kappa(x)$

procedure DPENCODE
  $bestEncoding[] \leftarrow String[|x| + 1]$
  $minEncoding[] \leftarrow Int[|x| + 1]$
  $opsEnding[] \leftarrow String[|x| + 1][]$
  $bestEncoding[0] = x[0]$
  $minEncoding[0] = 0$
  for i in $[1, |x|]$ **do**
    $ospEnding[i] \leftarrow \{s \in \Sigma', x[i - |s| : i] = s\}$
  end for
  for i in $[1, |x|]$ **do**
    $bestOp \leftarrow null$
    $m \leftarrow 0$
    for $s \in opEnd[i]$ **do**
      $t \leftarrow minE[i - |s|] + 1$
      if $bestOp = null$ or $t < m$ **then**
        $m \leftarrow t$
        $bestOp \leftarrow s$
      end if
    end for
    $minEncoding[i + 1] \leftarrow m$
    $bestEncoding[i + 1] \leftarrow bestEncoding[i - |bestOp|] + bestOp$
  end for
  return $bestEncoding[|x|]$
end procedure

3.3 State Update

When using classical PSR in online environments one typically updates the state vector every time a new observations is available. This eliminates the need for repeatedly transforming the initial state over the whole history of observations h when making predictions. In the case of M-PSRs, the dynamic programming algorithm for minimizing the length of string encodings provides a naturally convenient way to perform efficient state updates. To do so, we cache past state vectors along with their encoding length. When a new obser-

vation σ is read, we determine the encoding for the extended observation string $h\sigma$ with the same recurrence relation as in the previous section. Because this minimization is over $\{s \in \Sigma' \,|\, \exists p \in \Sigma^* \, ps = h\sigma\}$, one needs to cache at most $\max_{s \in \Sigma'} |s|$ state vectors and encoding lengths.

3.4 Greedy Selection of Multi-Step Observations

Algorithm 2 Base Selection Algorithm

INPUT: $Train$, Sub_M
OUTPUT: Σ'

procedure BASE SELECTION
  $\Sigma' \leftarrow \{s, s \in \sum\}$
  $bestEncoding \leftarrow null$
  for each obs in Train **do**
    $bestEncoding[obs] \leftarrow |obs|$
  end for
  $i \leftarrow 0$
  while $i < numOps$ **do**
    $bestOp \leftarrow null$
    $m \leftarrow 0$
    for each s $\in Sub_M$ **do**
      $c \leftarrow 0$
      for each obs in Train **do**
        $c \leftarrow c + DPEncode(obs, \Sigma \cup s) - bestEncoding[obs]$
      end for
      if $c < m$ **then**
        $bestOp \leftarrow s$
        $m \leftarrow c$
      end if
    end for
    $\Sigma' \leftarrow \Sigma' \cup bestOp$
    for each obs in Train **do**
      $bestEncoding[obs] \leftarrow DPEncode(obs, \Sigma')$
    end for
    $i \leftarrow i + 1$
  end while
  return Σ'
end procedure

Selecting multi-step transition sequences to build Σ' can be achieved with an adaptive greedy algorithm, depicted in Algorithm 2. A Σ' that correctly captures the frequent observations produced by a target environment should promote short encodings when coupled with the coding function κ described above. In practice, we want Σ' to contain 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, and thus interpret this approach as a data compression scheme.

In general terms, the algorithm works as follows. In a preprocessing step, we find all possible substrings in $\Sigma^\star$ that appear in the training dataset. For computational reasons this can be constrained only to the M most frequent substrings, where M is a parameter chosen by the user. Frequency of occurence is measured by number of training trajectories that contain a given substring. The construction of Σ' is then initialised by Σ and proceeds iteratively by adding a new multi-step symbol at each phase. A phase starts by evaluating all substrings in terms of how much reduction in the number of transition operators used to encode the whole training set would be achieved if the symbol was added to Σ'. The algorithm then adds to Σ' the multi-step symbol corresponding to the best substring, i.e. the one that would reduce the most the whole coding cost (with respect to κ). More formally, at the ith iteration the algorithm finds:

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

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.

4 Experiments

In this section, we assess the performance of PSRs and M-PSRs in various environments, with different model sizes, and learned from both large and small datasets. 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.

In all the experiments, an agent is positioned in a partially observable environment and navigates stochastically based on state to state transition probabilities. 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. We perform experiments in two environments: a Double Loop maze and a Pacman-style environment, depicted in Figure 2.

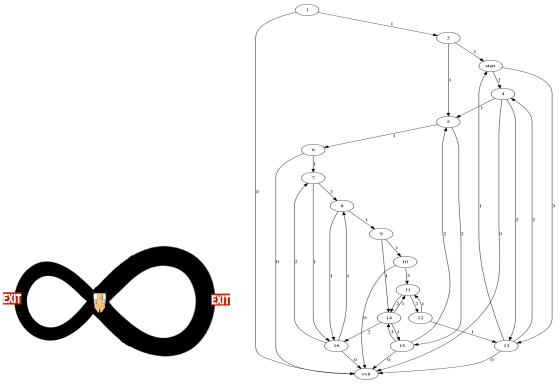

Figure 2: The Double Loop environment (left) and the Pacman Labyrinth (right)

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}.

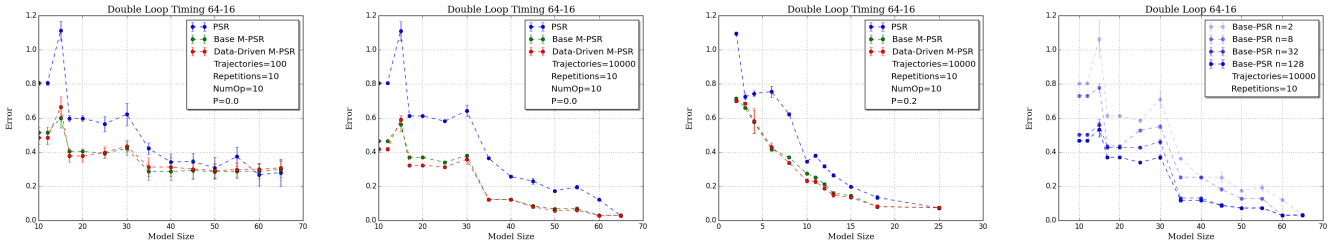

Figure 1: Results for the Double Loop of size 64-16 with (in order) a low amount of data, high amount of data and varying amount of noise (right two panels)

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 *et al.*, 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 in our dataset: $T = \{ps, \forall p \in \mathcal{P}, \forall s \in \mathcal{S}\}$

4.1 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 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.

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 when learning from 100 observation sequences, while the results in the next panel are obtained using 10000 observations. In both cases, the M-PSRs outperform the simple PSR for small model sizes. 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 $K = 7$.

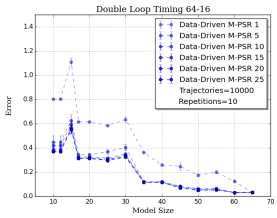

Figure 3: Effect of the number of operators on performance in 64-16 loop

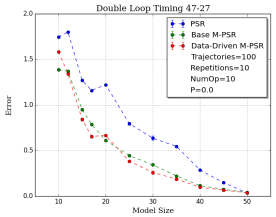

Figure 4: High Data Double Loop 47-27

In Figure 3, 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 are: $\{a, a^8, a^{24}, a^{32}, a^{72}\}$ which are closely tied to the environment's periodic structure.

In Figure 4, we plot the results of a 47-27 labyrinth. Here, because of the lengths of the loops, observations will not be as compactly expressed from the Base M-PSR. Again, M-PSRs outperform the standard PSR for reduced model sizes.

4.2 Pacman Labyrinth Timing

We now turn our attention to the environment on the right of Figure 2. Transitions to new states occur with equal probability. The weight $w(u, v)$ between states u and v corresponds

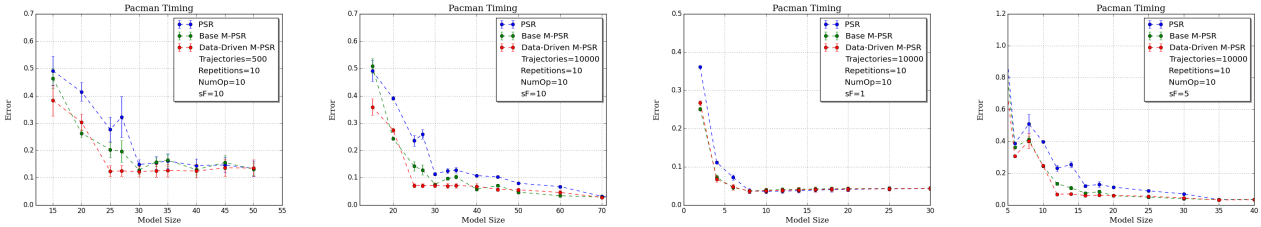

Figure 5: Results for the Pacman environment with (in order from left to right) low amount of data, high amount of data and stretch factor equal to 1 and 5 (right panels)

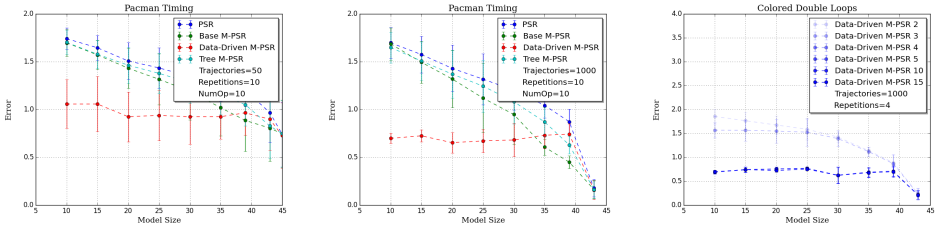

Figure 6: Results for Coloured Double loops 27-17, in low data regime (left, high data regime (middle) and as a function of different numbers of operators (right)

to the number of time steps taken to get from u to v. We add a parameter sF (stretch factor) which is used to scale all of the weights in the graph. In the left panels of Figure 5 we vary the number of observations used for learning. M-PSRs outperform the traditional PSR regardless of the amount of data. In the right two panels, we vary the stretch factor parameter, while keeping the size of the dataset fixed. We find that a higher value of sF provides increased improvement of the M-PSR relative to the standard PSR.

4.3 Multiple Observations: Coloured Loops

To test the case of multiple observations, we construct a Double Loop environment where the first loop has length $l_1 = 27$ and observations are green. 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:

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

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 learn the PSRS/M-PSRs in the left panels of Figure 6, and we find that M-PSRs perform far better, especially the Data-Driven M-PSR. In the right panel we vary the number of multi-step transition operators learned. The important operators learned are $\{g, b, g^{27}, b^{17}\}$, which is again very encouraging, as it reflects the structure of this particular environment.

5 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 *et al.*, 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 offer a computational advantage when performing conditional probability queries as they use far fewer matrices than regular PSRs. This can be very important for online applications, such as in planning environments.

We presented a comprehensive set of experiments 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 *et al.*, 2015] P.-L. Bacon, B. Balle, and D. Precup. Learning and planning with timing information in Markov Decision Processes. In *UAI*, 2015.

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

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

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

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

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

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

[Hamilton *et al.*, 2013] William L. Hamilton, Mahdi M. Fard, and Joelle Pineau. Modelling sparse dynamical systems with compressed predictive state representations. In *ICML*, 2013.

[Hsu *et al.*, 2009] D. Hsu, S.M. Kakade, and T. Zhang. A spectral algorithm for learning Hidden Markov Models. In *COLT*, 2009.

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

[Rosencrantz *et al.*, 2004] M. Rosencrantz, G. Gordon, and S. Thrun. Learning low dimensional predictive representations. In *ICML*, 2004.

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

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

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

[Zhang *et al.*, 2015] Chicheng Zhang, Jimin Song, Kamalika Chaudhuri, and Kevin Chen. Spectral learning of large structured hmms for comparative epigenomics. *Neural Information Processing Systems (NIPS)*, 2015.