TITLE GOES HERE

Anonymous Submission

Abstract

A standard problem in machine learning is making accurate predictions in partially observable environments. Spectral algorithms which learn PSRs offer statistical consistency, but often have trouble with computational efficiency as the models they learn are too large. In practice, one solves this computational issue by truncating the original model, eliminating the weakest states. Despite this, performance with reduced models is often quite poor. With this issue in mind, we develop a novel extension to PSRs which we call Multi-PSRs. In addition, we provide two algorithms for M-PSRs, one for learning parameters and the other for making effective queries. The M-PSR leverages structure in observations sequences and expresses queries more compactly. This extended model shares all the benefits of the traditional PSR but performs far better for smaller models in experiments. We perform experiments of robot exploration in labyrinth environments and cover both the single observation case (timing) and the multiple observation case. We pick these environments as PSRs have been used for planning applications on these kinds of environments in the past (CITE). In our experiments, we show that the improvement of M-PSRs hold for varying amounts of data, environment sizes, and number of observations symbols.

Introduction

Learning models of partially observable dynamical systems is very important in practice — several alternatives...

General algorithms are not designed to exploit frequen patterns/structure in sequences of observations to speed-up learning — but in practice with large observations and highly structured environments this might be necessary to achieve decent results

We propose a new model of predictive state representation for environments with discrete observations: the multi-step PSR (M-PSR)

We show how the standard spectral learning for PSR extends to M-PSR

Then we present a data-driven algorithm for selecting a particular M-PSR from data sampled from a structured partially observable environment

We evaluate the performance of our algorithms in an extensive collection of synthetic environments and conclude that...

The Muti-Step PSR

A linear *predictive state representation* (PSR) for an autonomous dynamical system with discrete observations is a tuple $\mathcal{A} = \langle \Sigma, \boldsymbol{\alpha}_\lambda, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma\in\Sigma}\rangle$ where: Σ is a finite set of possible observations, $\boldsymbol{\alpha}_\lambda, \boldsymbol{\alpha}_\infty \in \mathbb{R}^n$ 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) (**?**) computing a function given by the probability distribution of sequences of observations in a partially observable dynamical system with finite state. The function $f_\mathcal{A} : \Sigma^\star \to \mathbb{R}$ computed by $\mathcal{A}$ is given by

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

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 $\boldsymbol{\alpha}_\lambda$.

To define our model for multi-step PSR we basically 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 can be quite application-dependent, in order to reflect the particular patterns arising from different environments. However, we assume this 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 $\mathcal{A}' = \langle \Sigma, \Sigma', \kappa, \boldsymbol{\alpha}_\lambda, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma\in\Sigma'}\rangle$ containing a PSR with observations in Σ', together with the basic observations Σ and the corresponding coding function κ.

Borja: For now, this is enough

Examples

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

A PSR with a single observation $\Sigma = \{a\}$ can be used to measure the time – i.e. number of discrete time-steps – until a certain event happens (**?**). 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 whose lengths are powers of a fixed base. That is, given an integer $b > 0$, we build the set of multi-step observations as $\Sigma' = \{a, a^b, a^{b^2}, \ldots, a^{b^K}\}$ for some positive K. A natural choice of coding map in this case is the one that represents any length t 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 $0 \leq k \leq K-1$, and $t_K \geq 0$. With this decomposition we obtain the coding map $\kappa(a^t) = (a^{b^K})^{t_K}(a^{b^{K-1}})^{t_{K-1}} \cdots (a^b)^{t_1}(a)^{t_0}$. 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* (with base b and largest power K) for modelling distributions over time.

Borja: Re-wrote the presentation of the base M-PSR a little bit

For the multiple observation case one can also use The Base System. In this case $\Sigma' = \{\sigma^{2^k} \forall \sigma \in \sum, \forall k <= n\}$. For the encoding map κ we first split the string into sequences of a fixed symbol and then use the same encoding as for timing. For example $\kappa(a^5 b^3) = \{a^4, a, b^2, b\}$.

Another example for the multiple observation case is an M-PSR we will call the **Tree M-PSR**. For the Tree System, we set $\Sigma' = \{s \in \Sigma^*, len(s) <= L\}$. Here L is a parameter of choice. For the decoding map κ, we first split a string x as $x = x_1 x_2 ... x_n y$, where $|x_i| = L, \forall i <= n$ and $|y| = len(x) - (n \cdot L)$. With this we set $\kappa(x) = \{x_1, x_2, ..., x_n, y\}$.

The constructions of the M-PSRs above are not dependent on the environment. In our results section, we find that performance of M-PSRs depend heavily on how Σ' reflects the observations in one's environment. Thus, we develop an algorithm for choosing Σ'. In addition, we provide a κ which one can apply to any M-PSR and which delivers good experimental performance. Together, these yield another type of M-PSR, which we call a **Data-Driven M-PSR**.

Learning Algorithm for M-PSR

In this section, we describe a learning algorithm for M-PSR which combines the standard spectral algorithm for PSR (Boots, Siddiqi, and Gordon 2011) with a data-driven greedy algorithm for building an extended set of symbols Σ' containing frequent patterns that minimise a coding cost for a general choice of coding function κ.

Spectral Learning Algorithm

We extend (Boots, Siddiqi, and Gordon 2011) to M-PSR under the assumption that κ and Σ' are given.

Borja: Need to fill this. Probably copy-paste from the ODM paper will do.

Notation

Borja: We should move this into the two subsections below

Obs: A mapping from observation sequences to the number of occurrences of that sequence in one's dataset.

SubObs : all substrings of **Obs**.

Q: A query string (a string for which one wishes to determine the probability of).

bestE: A map from indices i of Q to the optimal encoding of Q[:i].

minE: A map from indices i of Q to $|bestE[i]|$

opEnd: A map from indices i of Q to the set of strings in Σ': $\{x \in \Sigma' s.t Q[i - x.length : i] == x\}$

numOps: The desired number of operators one wants in Σ'. I.e numOps = $|\Sigma'|$

A General Coding Function

Here we provide a dynamic programming algorithm which can serve as κ for any M-PSR. Given a query string Q, and a set of transition sequences Σ', the algorithm minimizes the number of sequences used in the partition $\kappa(Q)$. In other words, the algorithm minimizes $|\kappa(Q)|$. For the single observation case, the algorithm is equivalent to the coin change problem.

For a given string Q, the algorithm inductively computes the optimal string encoding for the prefix Q[:i]. It does so by minimizing over all $s \in \Sigma'$ which terminate at the index i of Q.

Greedy Selection of Multi-Step Observations

Here we present a greedy heuristic which learns the multi-step transition sequences Σ' from observation data. Having a Σ' which reflects the types of observations produces by one's system will allow of short encodings when coupled with our a dynamic programming encoding algorithm. In practice, this greedy algorithm will pick substrings from one's observation set 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 system's observations.

The algorithm evaluates substrings based on how much they reduce the number of transition operators used on one's observation data. The algorithm adds the best operator iteratively with Σ' initialized to Σ. More formally at the i'th iteration of the algorithm the following is computed: $min_{sub \in SubObs} \sum_{obs \in Obs} |\kappa(obs, \Sigma'_i \cup sub)|$. The algorithm terminates after the **numOps** iterations.

Experiments

We assess the performance of PSRs and different kinds of Multi-PSRs on labyrinth environments. We look to see how

Algorithm 1 Encoding Algorithm

procedure DPENCODE
  $bestE[] \leftarrow newString[len(Q)+1]$
  $minE[] \leftarrow newInt[len(Q)+1]$
  $opEnd[] \leftarrow newString[len(Q)+1][]$
  $bestEnd[0] = Q[0]$
  $minE[0] = 0$
  for i in range[1,Q.length] **do**
    $opEnd[i] \leftarrow \{s \in \Sigma', Q[i-len(s):i] == s\}$
  end for
  for i in range[1,Q.length] **do**
    $bestOp \leftarrow null$
    $m \leftarrow null$
    for $s \in opEnd[i]$ **do**
      $tempInt \leftarrow minE[i-len(s)]+1$
      if $m == null$ or $tempInt < m$ **then**
        $m \leftarrow temp$
        $bestOp \leftarrow s$
      end if
    end for
    $minE[i+1] \leftarrow m$
    $bestE[i+1] \leftarrow bestE[i-len(bestOp)] + bestOp$
  end for
  return $bestE[len(Q)]$
end procedure

Algorithm 2 Base Selection Algorithm

procedure BASE SELECTION
  $\Sigma' \leftarrow \{s, s \in \sum\}$
  $prevBestE \leftarrow null$
  for each obs in Obs **do**
    $prevBestEncoding[obs] \leftarrow len(obs)$
  end for
  $i \leftarrow 0$
  while $i < numOperators$ **do**
    $bestOp \leftarrow null$
    $bestImp \leftarrow null$
    for each s $\in ObsSub$ **do**
      $c \leftarrow 0$
      for each obs in Obs **do**
        $c \leftarrow c + DPEncode(obs) - prevBestE(obs)$
      end for
      if $c > bestImp$ **then**
        $bestOp \leftarrow observation$
        $bestImp \leftarrow c$
      end if
    end for
    $\Sigma' \leftarrow \Sigma' \cup bestOp$
    for each obs in Obs **do**
      $prevBestE \leftarrow DPEncode(obs, \Sigma')$
    end for
    $i \leftarrow i+1$
  end whilereturn Σ'
end procedure

performance varies as parameters are varied. Parameters include the model size, the number of observations used, and the type of environment. 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.

Obtaining Observation Sequences

In all the experiments we consider, an agent is positioned in a starting location and stochastically navigates the environment based on transition probabilities between states. When state-to-state transitions occur an observation symbol is produced. When the agent exits the labyrinth, we say the trajectory is finished, and we record the concatenation of the symbols produced. We call this concatenation the observation sequence for that trajectory.

Learning Implementation: Timing v.s Multiple Symbols

For the timing case, we construct our empirical hankel matrix by taking P,S = $\{\sigma^i, \forall i <= n\}$. The parameter n depends on the application. For Double Loop environments we set n = 150, while for the pacman labyrinth n = 600. For these choices of n, we verify that as the amount of data gets large the learned PSR with the true model size becomes increasingly close to the true model. For Base M-PSR, we set Σ' to be $\sigma^{2^k}, k <= 256$.

For multiple observations a slightly more complex approach is required to choose P and S. For prefixes P, we select the k most frequent prefixes from our observations set. For suffixes S, we take all suffixes that occur from our set of prefixes. We also require prefix completeness. That is if p' is a prefix of p $\in P$, then p' $\in P$. This heuristic for constructing empirical hankel matrices was given in previous work by [] and it showed that (). For **Base M-PSR**, we set Σ' to be $\{x^{2^k}, \forall x \in \Sigma', k <= 256\}$. For the **Tree M-PSR** we set L to 7.

Measuring Performance

For timing, the goal is to make predictions about how long the agent will survive the environment. One can also ask conditional queries such as how long the agent should expect to survive given that t seconds have elapsed

$$f(\sigma^m|\sigma^n) = \frac{\alpha_\lambda \cdot A(\kappa(\sigma^m)) \cdot \alpha_\infty}{\alpha_\lambda \cdot (I - A_\sigma)^{-1} \cdot \alpha_\infty}$$

The goal for the multiple observation labyrinths is to make predictions about seeing observation sequences. Conditional queries are also possible here

$$f(a^{m_1}b^{m_2}|a^{n_1}b^{n_2}) = \frac{\alpha_\lambda \cdot A(\kappa(a^{m_1}b^{m_2})) \cdot \alpha_\infty}{\alpha_\lambda \cdot A(\kappa(a^{n_1}b^{n_2})) \cdot \alpha_\infty}$$

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

$$||f - \hat{f}|| = \sqrt{\sum\nolimits_{x \in observations} (f(x) - \hat{f(x)})^2}$$

We use this norm because of a bound presented by [AUTHORS], which states that (). Here the function f denotes

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

Since the set of observations Σ^* is infinite, we compute approximations to this error norm, by fixing a set of strings T and summing over T. For the timing case, we take T to be the $\{\sigma^k, \forall k <= n\}$, while for the multiple observation case, we take all possible strings producible from the prefixes and suffixes in our dataset. That is, for the multiple observation case $T = \{ps, \forall p \in P, \forall s \in S\}$.

Double Loop Timing

For timing, we start by considering a double loop environment. The lengths of the loops correspond to the number of states in the loop. A trajectory begins with the agent starting at the intersection of the two loops. At the intersection of the two loops, the agent has a 50 percent chance of entering either loop. At intermediate states in the loops the agent moves to the next state in the loop with probability 1-P and remains in its current state with probability P. Here, P represents the self-transition probability for internal states. Exit states are located halfway between each loop. At an exit state, the agent has a 50 percent probability of exiting the environment.

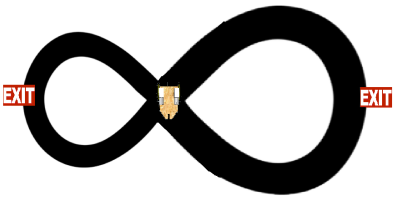

Figure 1: Double Loop Environment

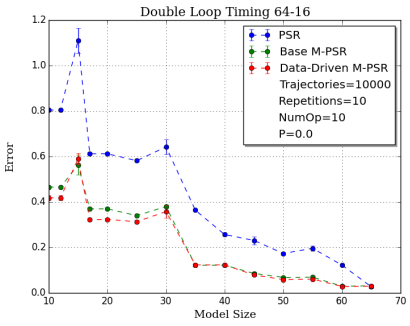

Figure 2: Double Loop 64-16

Number of Trajectories

Here, we vary the number of observations used in our dataset. PSRs/M-PSRs learned in Figure 4 use 100 observation sequences, while those in Figure 5 use 10000. In both cases the M-PSRs outperform the standard PSR for reduced model sizes.

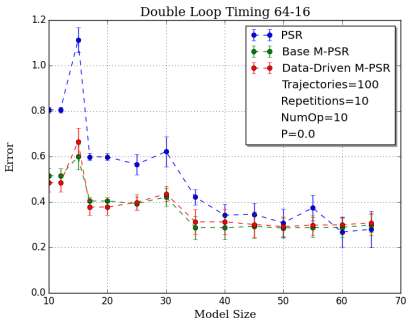

Figure 3: Double Loop 64-16

Noise: P

Next, we vary the self-transition probability P to simulate noise in an environment. Figure 5 is a 64-16 double loop with P=0.2, and Figure 6 is a 64-16 double loop with P=0. We find that the noisy loops are more compressible, but the performance is worse for higher models. Nevertheless, M-PSRs still significantly outperform the standard PSR for reduced model sizes.

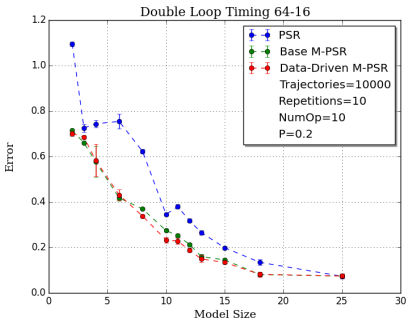

Figure 4: Double Loop 64-16

Loop Lengths

So far we have been using a 64-16 Double Loops. Here observations will come in low multiples of large powers of two. Intuitively, in this case the Base M-PSR should shine the most. In Figure 8, we plot the results of a 47-27 labyrinth where observations will not be so easily expressed from the Base M-PSR. Once again, M-PSRs outperform the standard PSR for reduced model sizes. In addition we see that the Data-Driven M-PSR does better than the Base M-PSR.

Large Labyrinth Timing

We proceed to work with a more complex labyrinth environment. Figure 10 shows it's graphical representation. We test

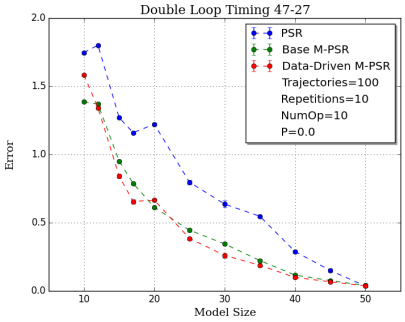

Figure 5: Double Loop 47-27

a larger environment as results in such a system would transfer to applications such as pacman. Transitions to new states occur with equal probability. The weight between transitions corresponds to the number of time steps. We add an additional parameter sF: stretchFactor, which multiplies all of the weights in the graph.

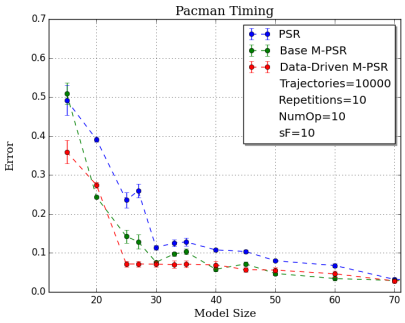

Figure 6: Pacman Labyrinth

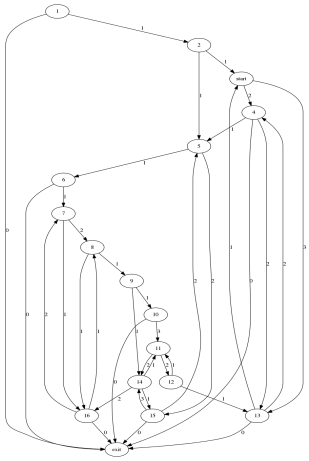

Figure 7: Graph of pacman

Number of Trajectories

In Figures 11 and 12 we vary the number of observations used for learning. First we note that performance of all models is significantly worse for less data. Nevertheless, M-PSRs outperform the traditional PSR regardless.

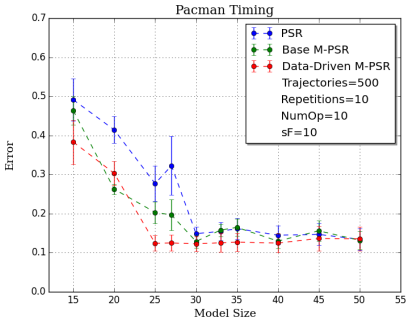

Figure 8: Pacman Labyrinth

Stretch Factor: sF

In Figures 13 and 14 we vary the stretch factor parameter. We find that a higher values of sF allow for increased improvement of the M-PSR relative to the performance of the standard PSR.

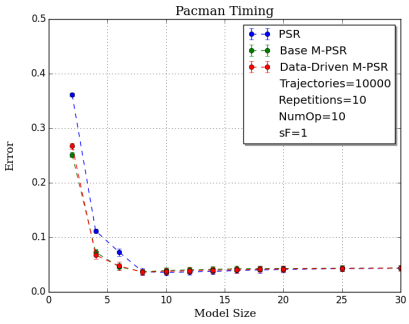

Figure 9: Stretch Factor: 1

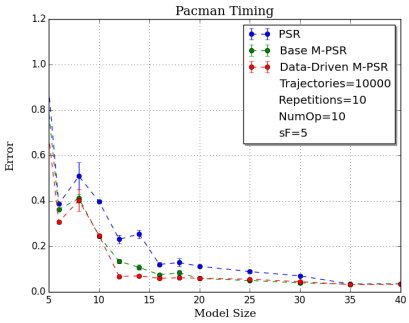

Figure 10: Stretch Factor: 5

Multiple Observations: Colored Loops

We now move to the multiple observation case. Here the Data-Driven M-PSRs really show their strength as observation sequences are more complex. We construct a Double Loop environment where one loop is green and the other is blue. The lengths of each loop are also varied, see Figure 4 and Figure 5. We fix the length of observations to be $TrajectoryLength := (len(loop1) + len(loop2)) * 3$. To build empirical estimates of probabilities we set $f(x) = \frac{\#occurances of x}{counts(s \in Obs, len(s) >= x)}$. This means that the PSRs will compute the probability of x occurring as a prefix.

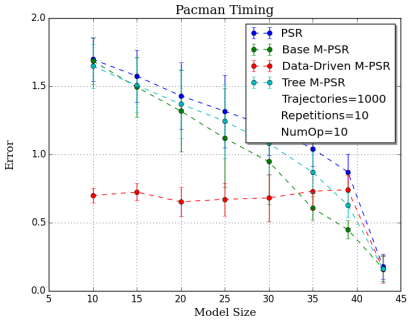

Figure 11: Colored Loops 27-17

Lucas: Removed picture of colored loops. Waste of space and leaves too much evidence of MS paint.

Number of Trajectories

As for the timing case, we vary the amount of data to learn PSRS/M-PSRs in Figures 15 and 16. Once again we find M-PSRs perform far better, especially the Data-Driven M-PSR. This makes sense as when complexity in observations increases only custom M-PSRs will express transitions compactly.

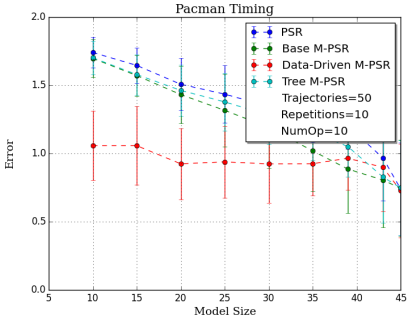

Figure 12: Colored Loops 27-17

Data Driven Sequences

For the double loop case the greedy approach learns multiples of the loop lengths which results in partitions which use fewer operators. As an example, for the 47-27 labyrinth the greedy heuristic picked the following operators: $\Sigma' = \{\sigma^{47}, \sigma^{27}, \sigma^{74}, \sigma^{94}...\}$. For Pacman, the learned strings are multiples of the stretch factor. For Colored Double Loops consistent operators in Σ' are $\{g^{27}, b^{17}, g^{27}b^{17}, b^{17}g^{27}\}$.

Conclusion

Acknowledgments

Funding and friends...

References

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