TITLE GOES HERE

Anonymous Submission

Abstract

Fancy Abstract...

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...

M-PSR: Definition and Learning

Multi-Step PSR

A linear *predictive state representation* for an autonomous dynamical system with discrete observations in a set Σ is a tuple $\mathcal{A} = \langle \Sigma, \boldsymbol{\alpha}_\lambda, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma\in\Sigma}\rangle$ where: BLA BLA.

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 $\mathcal{A}' = \langle \Sigma, \Sigma', \kappa, \boldsymbol{\alpha}_\lambda, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma\in\Sigma'}\rangle$.

Spectral Learning Algorithm

We extend (Boots, Siddiqi, and Gordon 2011)...

Examples of M-PSRs

In this section we go through examples of Multi-PSRs which are relevant to our experiment section.

For the timing case one can use a M-PSR which we call the **Base M-PSR**. For the Base M-PSR, Σ' is $\{a^{2^k} \forall k <= n\}$. For the encoding map we write an observation $a^m = a^{2^{n_1}} ... \cdot a^{2^{n_f}}$. With this equality we use the natural encoding map $\kappa(a^n) = \{a^{2^n_1}, ..., \{a^{2^n_f}\}$.

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 $\kappa(a^5b^3) = \{a^4, a, b^2, b\}$.

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 $x = x_1x_2...x_ny$, 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**.

Fully Data-Driven Learning

Overview

In this section, we provide two algorithms: the first being a general algorithm for κ and the second for choosing Σ' from Data. We present these algorithms together in the context of **Data-Driven M-PSRs** as the Data-Driven algorithm for choosing Σ' makes use of the algorithm for our choice of κ.

Notation

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': $\{x \in \Sigma' s.tQ[i - x.length : i] == x\}$

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

Learning the Encoding 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.

Learning Transition Operators

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

Algorithm 2 Base Selection Algorithm

procedure BASE SELECTION
  $\Sigma' \leftarrow \{s, s \in \sum\}$
  $bestOp \leftarrow null$
  $i \leftarrow 0$
  $bestImp \leftarrow null$
  $prevBestE \leftarrow null$
  for each obs in Obs **do**
    $prevBestEncoding[obs] \leftarrow obs.length$
  end for
  while $i < numOperators$ **do**
    for each s $\in ObsSub$ **do**
      $temp \leftarrow 0$
      for each obs in Obs **do**
        $temp \leftarrow temp + DPEncode(obs) - prevBestE(obs)$
      end for
      if temp>bestImp **then**
        $bestOp \leftarrow observation$
        $bestImp \leftarrow temp$
      end if
    end for
    $\Sigma' \leftarrow \Sigma' \cup bestOp$
    for each obs in Obs **do**
      $prevBestE \leftarrow DPEncode(obs, \Sigma')$
    end for
    $bestOp \leftarrow null$
    $bestImp \leftarrow null$
    $i \leftarrow i + 1$
  end whilereturn $BaseSystem$
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 including $\{\sigma^i, \forall i <= n\}$. With this choice, the empirical hankel matrix will be a nxn matrix with the prefixes and suffixes being the same.The parameter n depends on the application. For Double Loop environments we set n := 150, while for the pacman labyrinth n := 600. We verify for this choice of n 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 construct the empirical hankel matrix. 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 $f(\sigma^m|\sigma^n) = (eqns)$.

The goal for the multiple observation labyrinths is to make predictions about seeing observation sequences. Conditional queries are also possible here $f(a^{m_1}b^{m_2}|a^{n_1}b^{n_2}) = (eqns)$.

//DOUBLE CHECK THE ABOVE.

To measure the performance of a PSR/M-PSR we use the following norm: $||f - \hat{f}|| = \sqrt{\sum_{x \in observations}(f(x) - \hat{f(x)})^2}$. 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 $T = \{ps, \forall p \in P, \forall s \in S\}$.

FIXED DATASET NOTE: For a fixed dataset of observations, we compare the performance of a PSR and different M-PSRs and then plot the average error over the 10 datasets.

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.

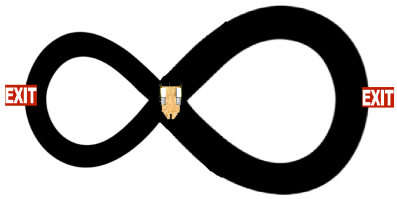

Figure 1: Double Loop Environment

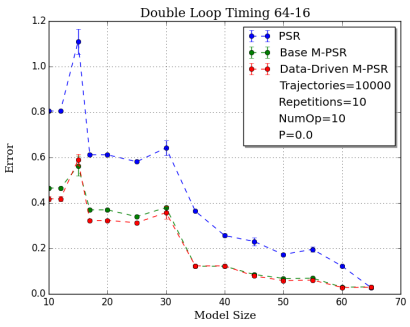

Figure 2: Double Loop 64-16

Amount of Data

Here, we vary the number of observations used in our dataset. PSRs/M-PSRs learned in Figure 4 use 100 obser-

vation sequences, while those in Figure 5 use 10000. In both cases the M-PSRs outperform the standard PSR for reduced model sizes.

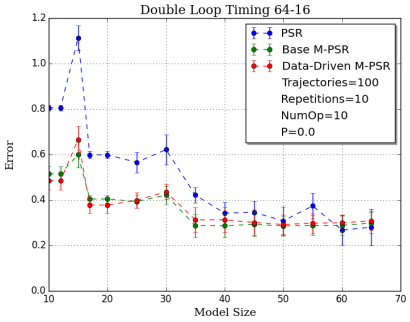

Figure 3: Double Loop 64-16

Noise: Parameter 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.

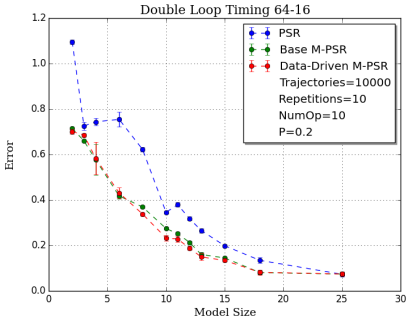

Figure 4: Double Loop 64-16

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

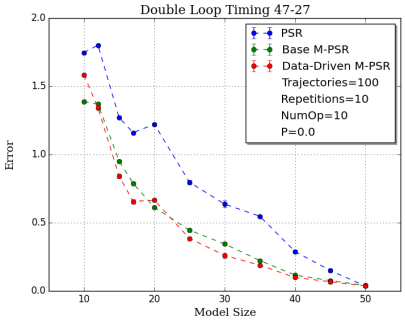

Figure 5: Double Loop 47-27

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.

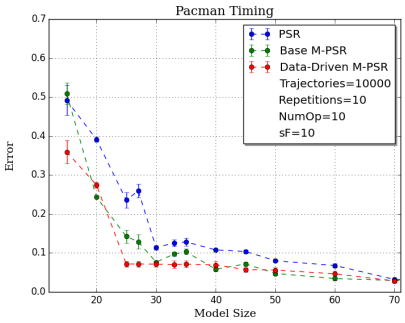

Figure 6: Pacman Labyrinth

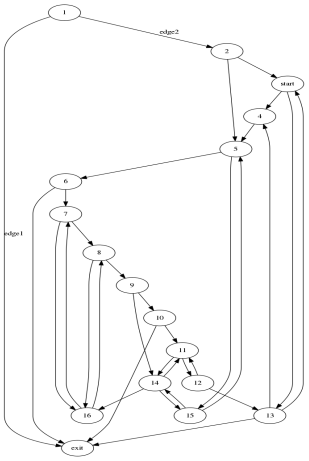

Figure 7: Graph of pacman

Amount of Data

In Figures 11 and 12 we vary the number of observations used to learn PSRs/M-PSRs. We find that ().

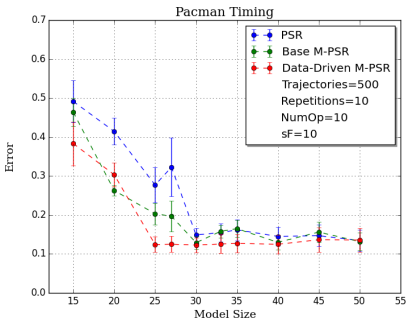

Figure 8: Pacman Labyrinth

StretchFactor

In Figures 13 and 14 we vary the stretchFactor parameter. We find that ().

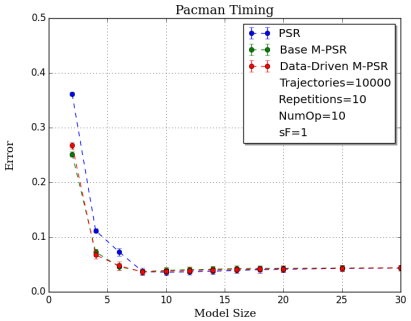

Figure 9: Pacman Labyrinth

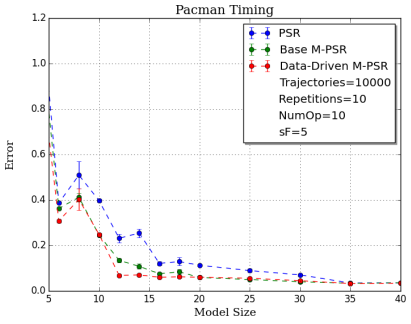

Figure 10: Pacman Labyrinth

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 $f(x) = (pocc(x)/numstringslength >= x$. This means that the PSRs will compute the probability of a prefix occurring.

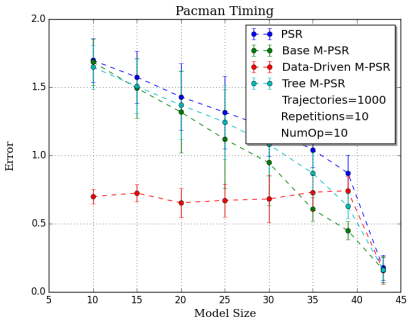

Figure 11: Colored Loops 27-17

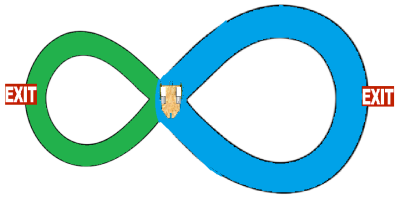

Figure 12: Colored Loops Environment

Amount of Data

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 ().

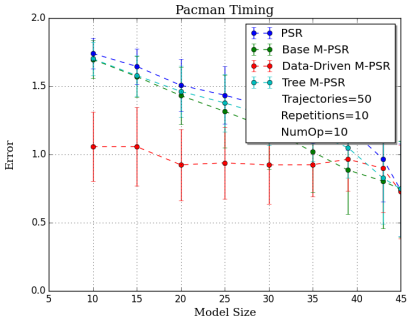

Figure 13: Colored Loops 27-17

Data Driven Sequences

For the double loop case the data-driven 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, $\Sigma' = \{g^{27}, b^{17}, g^{27}b^{17}, \}$ The top 7 of 10 operators are usually learned consistently, while the other strings vary slightly dependent on the generated dataset.

Discussion

some discussion of results...

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*.