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

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

For the timing case one can use a M-PSR which we call the **Base System**. For this type of M-PSR, $\sum'$ will consist of $\{a^{2^k} \forall k <= n\}$. To describe the encoding map we write an observation $a^m = a^{2^{n_1}} ... \cdot a^{2^{n_f}}$ With this equality a natural encoding map κ which is natural to use is: $\kappa(a^n) = \{a^{2^n_1}, ..., \{a^{2^n_f}\}$.

For the multiple observation case one can also use The Base System. In this case $\sum' = \{\sigma^{2^k} \forall \sigma \in \sum, \forall k <= n\}$. 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. The Base System for multiple observations clearly aims at environments where observation symbols come in streaks.

Another example for the multiple observation case is an M-PSR we will call the **Tree System**. For the Tree System, we set $\sum' =$ the set of all possible strings of length $<= L$. For the decoding map κ, we first split a string x into $x_1x_2...x_ny$, where $xi\ \forall i <= n$ have length L and y has length $len(x) - n \times L$. With this we set $\kappa(x) = \{x_1, x_2, ..., y\}$.

The the construction of the above multi-PSRs is not dependent on the environment. In practice, if one would like to learn a M-PSR which has the best performance, namely a M-PSR whose parameters best reflect the types of observations seen from one's environment.

Fully Data-Driven Learning

Learning Transition Operators

Here we present a greedy heuristic which learns our transition sequences $\sum'$ from observation data. We start with some notation. Let ...

ALGORITHM CODE:

Learning the encoding function

Here we provide a dynamic programming algorithm which can serve as $kappa$ for any M-PSR. Given a query string Q, and a set of transition sequences $\sum'$, the algorithm minimizes the number of sequences used in the partition. In other words, the algorithm minimizes $|k(Q)|$.

ALGORITHM CODE:

Experiments

We assess the performance of the Base System on labyrinth environments. The robot is positioned in a starting location and it stochastically navigates the environment. We split the experiments into two cases. The first is the case of timing, where $\sum = \{\sigma\}$ and the second is with multiple observations. 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 x seconds have elapsed. For multiple observations we place the agent in the environment, let it roam for a fixed time length and then remove the agent. The goal for the multiple observation labyrinth will be to make predictions about seeing observation sequences. In both cases, we analyse the performance for PSRs of different model sizes with a fixed observation set. For each Base System PSR, we include all powers of 2 up to 256. For the heuristic based PSRs we learn 10 operators from the observations with the greedy algorithm described in (). To measure the performance of a PSR we use the following norm: $||f - \hat{f}|| = \sqrt{\sum_{x \in observations}(f(x) - f(\hat{x}))^2}$. We use this norm because of a bound presented by [AUTHORS], which states that ().

Learning PSRs for Timing

For the timing case, we construct our empirical hankel matrix by including $\sigma^i i <= n$. 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 to be 300, while for pacMan it was 600. The important property that needs to be satisfied when choosing the parameter n is that enough observations are captured to learn a good model. Something here: $\lim_{\to 2} f(x) = 5$.

Learning PSRs for Multiple Observations

For multiple observations a slightly more complex approach is required to construct the empirical hankel matrix. For prefixes, we select the k most frequent prefixes from our observations set. For suffixes we take all suffixes that occur from our set of prefixes. This heuristic was given in previous work by [] and it showed that ().

Double Loop Timing

For timing, we start by considering a double loop environment. The agent starts at the intersection of the two loops. At the intersection, the agent has a 50 percent chance of entering either loop. Exit states are located halfway between each loop. At an exit state, the agent has a 50 percent probability of exiting the environment. Observations are generated by placing the agent in the environment: σ^{100} corresponds to the agent surviving 100 time units. In figure 1, we learn a PSRs with 10000 observation sequences. We generate 10 PSRs and average their performance.

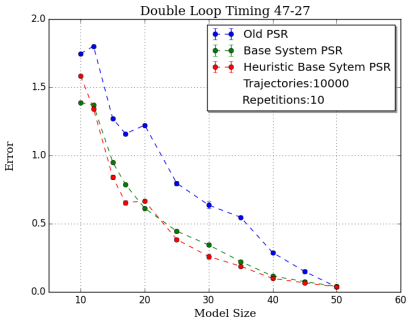

Figure 1: Double Loop Environment

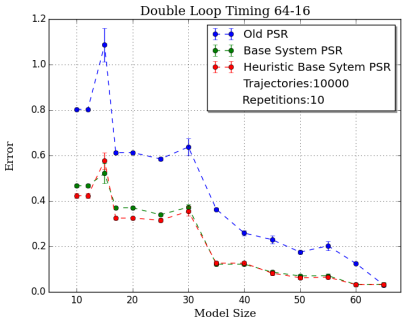

Figure 2: Double Loop Environment

Pacman Timing

We proceed to work with timing in a Pacman environment. The transition structure of this environment is shown in Figure 2. Edge weights vary from 1 to 3 and are stretched by a parmater we call the stretch factor which we initialize at 10. Again we use 10000 observation sequences per PSR and compute the average of 10 PSRs.

Timing - Results

As is demonstrated in figure 3, learning longer transitions provides a significant improvement over the standard for truncated models. The heuristic based approach performs slightly better than the powers of two method. For the double loop case the heuristic approach learns multiples of the loop

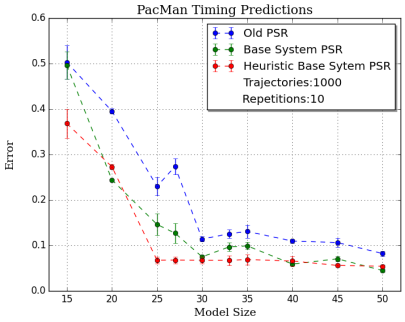

Figure 3: Double Loop Environment

lengths which results in partitions which use fewer operators. For Pacman, the operators that are learned are various multiples of the stretch factor, which once again shows that the greedy heuristic is effective.

Multiple Observations

To test whether the Base System would translate to multiple observations we construct a double loop environment where one loop is green and the other is blue. The lengths of each loop are also varied. We fix the length of observations to be $loop1 + loop2 * 3$. To build empirical estimates of probabilities we set f(x)=prefix-occ(x)/num-strings-length¿=x.

Multiple Observations Results

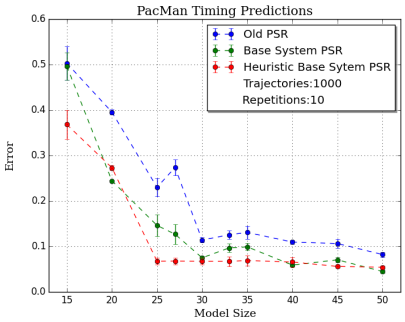

Figure 4: Double Loop Environment

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*.