Discovering and characterizing Hidden Variables

Soumi Ray and Tim Oates
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, MD 21250

Abstract

Theoretical entities are aspects of the world that cannot be sensed directly but that nevertheless are causally relevant. Scientifc inquiry has uncovered many such entities, such as black holes and dark matter. We claim that theoretical entities are important for the development of concepts within the lifetime of an individual, and present a novel neural network architecture that solves three problems related to theoretical entities: (1) discovering that they exist, (2) determining their number, and (3) computing their values. Experiments show the utility of the proposed approach using a discrete time dynamical system in which some of the state variables are hidden, and sensor data obtained from the camera of a mobile robot in which the sizes and locations of ob jects in the visual eld are observed but their sizes and locations (distances) in the three-dimensional world are not.

Introduction

Humans, like robots, have limited sensory access to the physical world. Despite this fact, thousands of years of scientific inquiry have uncovered much hidden structure governing the behavior and appearance of the world, along the way proposing a vast array of entities that we cannot see, hear, taste, smell, or touch. When Gregor Mendel discovered genes in the middle of the 19th century, he couldn't experience them in the same way he could experience, say, the smell of a rose or the color red. However, genes have causal power that manifests itself in ways that can be sensed directly. For Mendel, one such manifestation was the color of the peas of the pea plants that he bred with one another. Whether a plant would have yellow or green peas could not be predicted accurately based solely on observable properties of the parent plants. The desire to explain this apparent non-determinism led Mendel to posit the existence of *a causally efficacious entity of the world that could not be sensed directly*, i.e., genes. Such entities are called *theoretical entities*.

Theoretical entities are of fundamental importance to the development of human knowledge. The history of science is replete with reliance on and corroboration of the existence of theoretical entities, like genes, atoms, gravity, tectonic plates, germs, dark matter, electricity, and black holes. No one has ever seen a black hole, yet most physicists believe they exist because black holes accurately explain a wide array of observational data. Human knowledge would be limited indeed were we restricted to only represent and reason about things that we can see, hear, taste, smell, or touch.

This paper presents a novel neural network architecture for discovering hidden variables in time series data. The architecture is able to discover the existence of hidden variables, determine their number, and estimate their values. Empirical results are presented for a discrete time dynamical system and data gathered from a robots interactions with objects.

Background

McCallum (1996) did early work on hidden states in the context of reinforcement learning, where hidden states are typically ignored and traditional reinforcement learning methods are applied in fixed-memory agents. In other cases an agent's current percepts are augmented with history information. The problem in this situation is one of memory and storage. McCallum proposed a method called *instance-based state identification*, where raw data from previous experiences are stored directly. The simplest instance-based technique is the *nearest sequence memory* which is based on k-nearest neighbors. This technique, though simple, improved the performance of learning and took fewer training steps for learning. The main disadvantage of this technique is that, though it learns good policies quickly, it does not always learn the optimal policy.

Significant research in the recent past has focused on the problem of learning Bayesian Networks (BN) from data. Elidan *et al.* (2000) presents an algorithm for discovering hidden variables in Bayesian Networks by looking for cliques in network structures learned assuming all variables are observable. When a hidden variable is known to exist, they introduce it into the network and apply known BN learning algorithms. First, using the standard Bayesian model selection algorithm, a structure over the observed variables is learned. Then the structure is searched for sub-structures which they call semi-cliques. A hidden variable is then introduced to

break this clique and then learning is continued based on that new structure.

Similarly, work on planning under uncertainty using, for example, the Partially Observable Markov Decision Process (POMDP) framework assumes knowledge of the number of underlying hidden states (Kaelbling, Littman, & Cassandra, 1996). The agent whose world is characterized by the POMDP does not have access to the state that it actually occupies. Rather, the agent maintains a belief state, or probability distribution over states that it might be occupying. This belief state is Markovian, meaning that no additional information from the past would help increase the expected reward of the agent. Again, the goal of this work is not to discover the existence of a hidden state, but to behave optimally given knowledge of the existence of hidden state. More recently, Littman, Sutton, & Singh (2001) showed that dynamical systems can be represented using Predictive State Representations (PSRs), or multi-step, action-conditional predictions of future observations, and that every POMDP has an equivalent PSR. PSRs can look both at the past and summarize want happened and can also look to the future and predict what will happen. PSR is a vector of tests Rivest & Schapire (1994) which stores the predictions for a selected set of action-observation sequences. Holmes & Isbell (2006) showed that the unobserved or hidden states can be fully captured by a finite history based representation called a looping prediction suffix tree (PST). They focus on cases of POMDPs where the underlying transition and observation functions are deterministic.

Latent variables are important in the Psychology and Social Science research. Bollen (2002) described three definitions of latent variables: *local independence*, *expected value true score*, and *non-deteministic functions of observed variables* and introduced a new notion of latent variables called "*sample generalizations*". Latent variables can be defined *non-formally* or *formally*. Non-formally latent variables can be considered as *hypothetical variables* or unobserved variables as a data reduction device. Hypothetical variables are variables considered imaginary, i.e. not existing in the real world. Unobservable variables are impossible to be measured. The third non-formal definition of latent variables defines them as a data reduction device that can be used to describe a number of variables by a small number of factors.

One of the most common and popular formal definitions of latent variables is the *local independence* definition (Lord 1953, Lazarsfeld 1959, McDonald 1981, Bartholomew 1987, Hambleton et al. 1991). It means that the observed variables are associated with each other because of one or more latent variables. If the latent variables are known and are held constant then the observed variables become independent. This can be defined more formally:

$$P(Y_1, Y_2, \ldots, Y_k|N) = P(Y_1|N)P(Y_2|N)\ldots P(Y_k|N) \quad (1)$$

where $Y_1, Y_2, \ldots, Y_k$ are random observed variables and N is a vector of latent variables.

The next formal definition of latent variables defines a *true score*. The true score is calculated as the expected value of the observed variable for a particular object. Another definition of latent variables is that latent variables are non-deterministic functions of the observed variables, that is, they cannot be expressed as a function of the observed variables (Bentler 1982). It might be possible to predict a value of the latent variable but it is not possible to exactly predict the value of the latent variable based on the observed variables. The definition introduced by Bollen for latent variables is *sample realization*. He said that a latent variable is a random (or nonrandom) variable for which there is no sample realization for some observations, that is, there are no values for those observations. Observed variables contain sample realization while latent variables do not.

Some of the useful properties of latent variables were also discussed in Bollen's paper. A latent variable is denoted as *a posteriori* if it derived from a data analysis. On the other hand, *a priori* latent variables are hypothesized before the data analysis is done. Another property of latent variables can be understood by finding if they are affected by the observed variables or observed variables are the effects of the latent variables.

We are interested in finding hidden variables in time series data in a partially observable environment. We are not only interested in discovering hidden variables but also find the number of hidden variables in a given situation.

Method

We have designed a novel neural network architecture for the discovery and quantification of hidden variables. Our neural network architecture is comprised of two linked networks, the original network (O net) and the latent network (L net), as shown in Figure 1. We call this network the *LO net*. History values of the observed process are input into each component network, and the output of the L net is also an input of the O net.

Consider the problem of predicting the value of a variable x at time $t+1$, given information up to time t. The current and previous two values are provided as inputs to both the original and the latent network and the next value is predicted as shown in Figure 1. The input to the latent network is the current and previous two values at all times. The input to the original network is initially all three values, but with more learning the history values are dropped sequentially. This is done to give more responsibility to the latent network. The latent network can learn to output an approximation to the hidden variables. The network is trained using gradient descent backpropagation.

Since the latent network is not provided with any example outputs, the only way it learns is from the errors back-propagated to it. We want the latent network to learn the value of the hidden variable. The idea behind dropping the history inputs from the original network

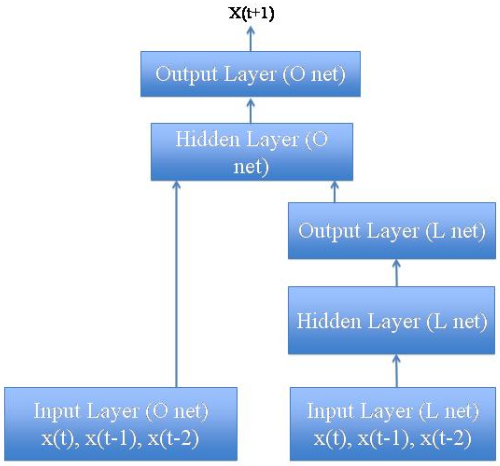

Figure 1: Our Network Architecture.

as learning progresses is to make the output from the latent network a crucial input to the original net. Hence the performance of the whole network will depend on what the latent network is learning and we expect the latent network to approximate the hidden variable. In our example we have taken a history of size two, i.e., the two previous observations. This method can work for smaller or larger history sizes. The history size will vary for different domains.

A theorem by Takens (Takens, 1981) states that for discrete-time deterministic dynamical systems of n variables, it is possible to exactly recover the topology of the system by treating each window of $2n$ consecutive values of just one variable as a state. This provides the basis for heuristic ideas in using history values to evaluate processes with hidden variables. The use of the neural network provides us the flexibility to mimic the transformation in Taken's theorem without worrying about its particular functional form. The neural network architecture thus provides a promising approach for estimating the true underlying system including hidden variables.

In our implementation the network has each of its layers' weights and biases initialized with the Nguyen-Widrow layer initialization method. The Nguyen-Widrow method generates initial weight and bias values for a layer so that the active regions of the layer's neurons are distributed approximately evenly over the input space. The values contain a degree of randomness, so they are not the same each time this function is called. The training function used to update the weight and bias values in the network is gradient descent with momentum and adaptive learning rate backpropagation. The parameter lr indicates the learning rate, similar to simple gradient descent. The parameter mc is the momentum constant that defines the amount of momentum. mc is set between 0 (no momentum) and values close to 1 (high momentum). A momentum constant of 1 results in a network that is completely insensitive to the local gradient and, therefore, does not learn properly. The momentum constant (mc) used was 0.9. The learning rate (lr) we have chosen is 0.01. For each epoch, if performance decreases toward the goal, then the learning rate is increased by the factor lr-inc (1.05). If performance increases by more than the factor max-perf-inc (1.04), the learning rate is adjusted by the factor lr-dec (0.7) and the change, which increased the performance, is not made. A transfer function is used to calculate the i^{th} layer's output, given the layer's net input, during simulation and training. Backpropagation is used to calculate derivatives of performance ($perf$) with respect to the weight and bias variables X. The network's performance is measured according to the mean squared error. Each variable is adjusted according to gradient descent with momentum given in Eq 2,

$$d\,\mathrm{X} = mc * d\,\mathrm{Xprev} + lr * (1 - mc) * d\,\mathrm{perf}/d\,\mathrm{X} \quad (2)$$

where $d\,\mathrm{Xprev}$ is the previous change to the weight or bias. The transfer function used to calculate the hidden layer's output is the *tan-sigmoid* transfer function and the output layers use a *linear* transfer function.

Robot Data

Real world data was provided for this project by a surveyor SRV-1 Blackfin robot. The robot consists of a camera mounted on a pair of tank style treads that can be controlled remotely by a user interface on a laptop. The robot was placed in a fairly uniform environment (in this case the UMBC Department of Computer Science lobby) and driven by a human around a target. The targets consist of several brightly colored boxes, easily distinguishable from the surrounding environment by our image processing software. The surveyor would approach a target from different angles, keeping it in view the entire time for some trials, and for others occasionally facing different directions. Each frame transmitted from the surveyor's camera was recorded for later processing. The computation done on these frames consisted of counting the number of pixels that were present in a certain color range (giving us the surveyor's perception of the size of the box), and the centroid of the pixels of that color. Before each experiment, the color range was calibrated to avoid the surveyor mistaking other things for its target.

The absolute position of the robot in relation to its target was calculated by a camera hanging above the area in which the tests were being performed. The surveyor was tagged in its center with another unique color, and the camera was able to observe the position of the surveyor in relation to the box. This data was used to reconstruct the path of the robot, which was fitted across the data taken from the surveyor's camera in order to give us an approximation of the surveyor's position at each point.

The robot's vision system extracts the following information for a given box:

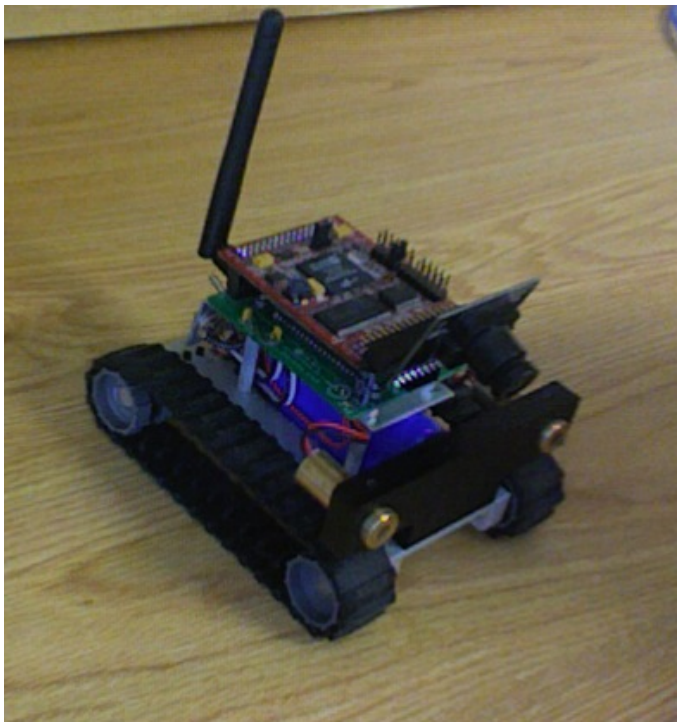

Figure 2: SRV-1 Blackfin Robot

s_i - the size of the object in the image plane.

x_i - the x coordinate of the object in the image plane

y_i - the y coordinate of the object in the image plane

Each object has an objective size s_0 and objective location (x_0,y_0,z_0), relative to the origin of the coordinate frame.

In general, if the camera moves in the 3 dimensional space with translational velocity $v = (v_x, v_y, v_z)$ and rotational velocity $\omega = (\omega_x, \omega_y, \omega_z)$, then the velocity of a point in the image plane can be expressed as follows:

$$\begin{aligned} \dot{x}_i &= t_x + r_x \\ \dot{y}_i &= t_y + r_y \end{aligned}$$

where,

$$\begin{aligned} t_x &= (-v_x + v_z x_i)/z_0 \\ t_y &= (-v_y + v_z y_i)/z_0 \\ r_x &= \omega_x x_i y_i - \omega_y(1 + x_i^2) + \omega_z y_i \\ r_y &= \omega_x(1 + y_i^2) - \omega_y x_i y_i - \omega_z x_i \end{aligned}$$

For our robot, the translational velocities v_x and v_y and rotational velocities w_x and w_z are physically constrained to be zero. So the equations for $\dot{x}_i$ and $\dot{y}_i$ are:

$$\dot{x}_i = v_z x_i/z_0 + w_y(1 + x_i^2) \quad (3)$$

$$\dot{y}_i = v_z y_i/z_0 + w_y x_i y_i \quad (4)$$

where v_z and w_y are constants.

The position of the image plane at each time step is then given by:

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_t \\ y_{t+1} &= y_t + \dot{y}_t \end{aligned}$$

Note that all the quantities required to predict the next value of x_{t+1} and y_{t+1} are observable except z_0,the distance of the robot from the box.

The perceived size of an object s_i depends on the objective size s_0 and the distance z_0 of the object as follows:

$$s_i = s_0/z_0^2 \quad (5)$$

The robot's perception of the size of the target thus changes with the distance from the target, though the target itself is of constant size. The quantities s_0 and z_0 are not observable, so s_i cannot be directly estimated. However, since s_0 is constant and our perspective projection is planar, we have a simpler situation where s_i changes only with z_0.

Experiments

This section presents the results of using the LO net architecture to predict future output based on history with the robot data. A robot collects data by going back and forth looking at an object. It records the x and y coordinates of the box and the size of the box in its vision. The network is trained for 400 epochs since around that time the MSE converges to a very low value. The plots show the MSE of the last 150 epochs. Initially the MSE is very high (around a few hundred) but it drops rapidly to around 50 in just first 7 or 10 epochs. Figure 3 plots the MSE of variable x. The solid line shows the performance when the current value of x (x_t) is fed and the the next value of x (x_{t+1}) is predicted, using only the original network. The dashed line shows the performance when the current and the previous two x values (x_t, x_{t-1}, x_{t-2}) are fed to the original network and the next value of x x_{t+1} is predicted. The dash-dot line and the dotted lines show the performance with one and two latent networks respectively. Initially for the first 100 epochs the current and the previous two x values (x_t, x_{t-1}, x_{t-2}) are fed to the original and latent networks. The output of the latent networks are also given as an input to the original network as shown in figure 1. and the next value of x (x_{t+1}) is predicted. In the next 100 epochs one history value x_{t-2} is dropped from the original network and training is continued. In the last 200 epochs the original network is fed with only the current value of x (x_t) and the output from the latent network. All the four figures plot the MSE versus the number of epochs. In the first case there is only one network and the input is just the current value. The second case is where there is also just one network but there are three inputs, the current and two previous inputs. The third and the fourth case show the results of the LO net architecture. In the third case there is one latent network along with the original network. In the fourth there are two latent networks. The x-axis

plots the number of iterations and the y-axis plots the mean square error (MSE) in the following three figures.

In figure 3 the performance of the network with three

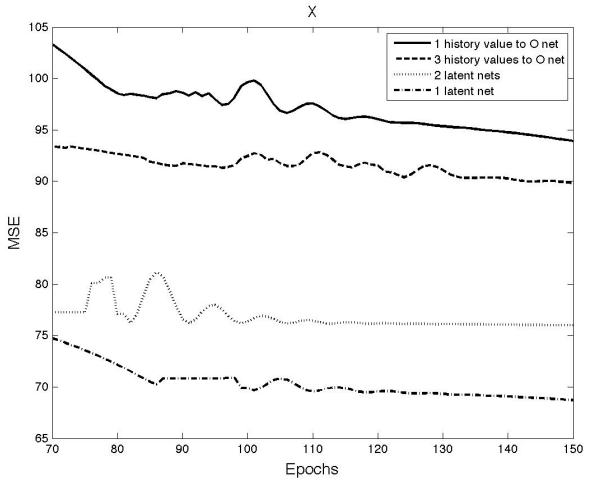

Figure 3: Performance curve for X.

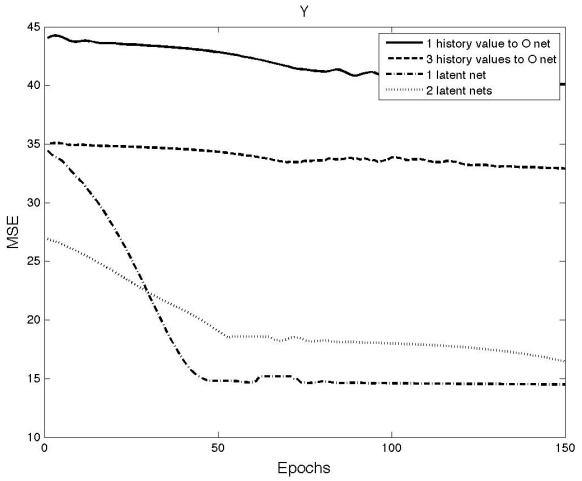

Figure 4: Performance curve for Y.

history values can be seen to be better than the performance with just the current value. The one LO net performs best in this case. It performs better than the two latent network architecture also. From equation 5 it is clear that there is one value which is unobservable for the prediction of x_{t+1} which is the distance of the robot from the box. While trying to predict the the next value of x with just the previous value of x one variable is hidden to x on which it is dependent. The output from the latent network in the LO net architecture provides input to the original input that improves its performance. The latent network posits the presence of a hidden variable. It approximately learns the value of the hidden variable. Initially three history values are provided as input to the original network but with more learning history values are dropped and so the input from the latent network becomes more important. We propose that the backpropagation algorithm updates the weights of the latent network in such a way so as to approximate the hidden variable. Similar results can be seen in the case of predicting y_{t+1} and s_{t+1}. The one latent network architecture improves the performance of learning in all the three cases. Adding a second latent network in these cases reduces the performance.There are two unobservable values for predicting the size of the box — the distance of the robot from the box and the actual size of the box. Since the actual size is constant the perceived size of the box changes only when the distance changes. So the latent network comes up with just one hidden variable in this case.

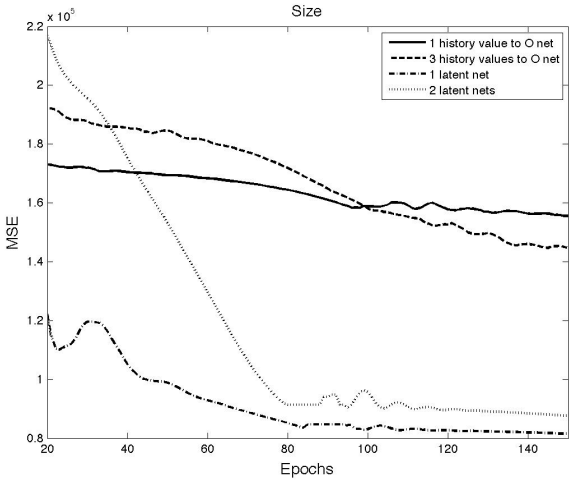

Figure 5: Performance curve for size.

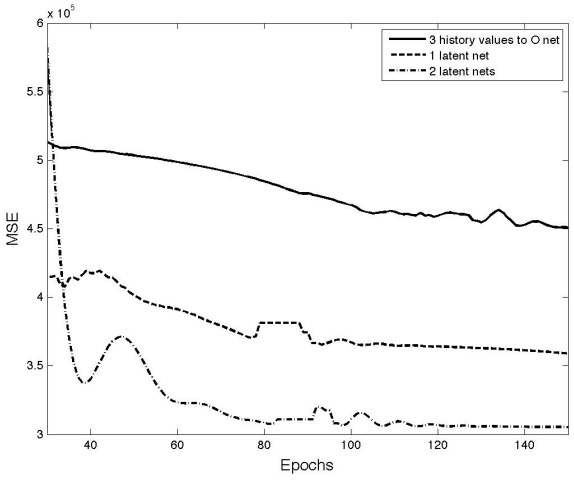

Figure 6: Performance curve for size with two boxes.

The next experiments are performed on data where there are two boxes in the vision of the robot. The architecture with two latent networks performs best when predicting the size of the box as seen in figure 6. The size of a box depends on the actual size of the box and the distance of the box from the robot. When there are two boxes the actual size is no more constant. So when predicting the next value of the size perceived by the robot the two hidden variables are the size and the distance. Adding a third latent network again reduces the performance of learning.

From these results we conclude that the performance of prediction of the future values can be improved by using the LO net architecture. Not only does it estimate the existence of hidden variables but it also gives an estimate of the number of hidden variables. For example x_{t+1} and y_{t+1} depend only on one unobservable variable, so one latent network does a better job than two latent networks. In the two latent network case the extra input from the second latent net reduced the performance. While predicting the future values of s_{t+1} with boxes in the robot's view which depends on two unobservable variables, two latent nets did a better job than one. The network architecture was able to predict two hidden variables.

Figure 7 show the outputs of the latent networks from the three experiments with one latent network while trying to predict the next values of x, y and size with one box in the robot's vision. All the three latent networks try to approximate one variable which is hidden, the distance of the robot from the box. It can be seen that the outputs from the latent networks are somewhat correlated.

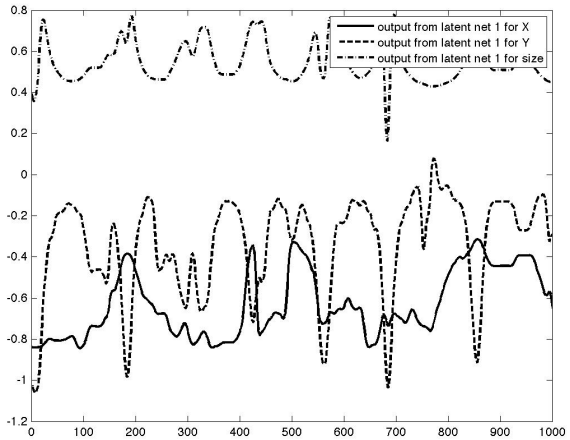

Figure 7: Comparison of the output values from the latent nets.

This neural network architecture can find the existence of hidden variables. The number of hidden variables can be found by iteratively adding latent networks to the original network until adding a new latent network does not significantly help. Our next experiments will be on large domains to see how this method scales when the number of hidden variables increases.

Conclusion

We presented a novel neural network architecture that solves three problems related to theoretical entities: (1) discovering that they exist, (2) determining their number, and (3) computing their values. Experiments showed the utility of the proposed approach using a discrete time dynamical system in which some of the state variables are hidden, and sensor data obtained from the camera of a mobile robot in which the sizes and locations of objects in the visual field are observed but their sizes and locations (distances) in the three-dimensional world are not.

Acknowledgement

We would like to thank Max Morawski for running the experiments with the robot and providing the experimental data.

References

Bollen, K. A. 2002. Latent variables in psychology and the social sciences. *Annual Review of Psychology* 53(1):605–634.

Elidan, G.; Lotner, N.; Friedman, N.; and Koller, D. 2000. Discovering hidden variables: A structure-based approach. In *NIPS*, 479–485.

Holmes, M. P., and Isbell, Jr., C. L. 2006. Looping suffix tree-based inference of partially observable hidden state. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, 409–416. New York, NY, USA: ACM.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1996. Planning and acting in partially observable stochastic domains. Technical Report CS-96-08.

Littman, M.; Sutton, R.; and Singh, S. 2001. Predictive representations of state.

McCallum, A. 1996. Hidden state and reinforcement learning with instance-based state identification.

Rivest, R. L., and Schapire, R. E. 1994. Diversity-based inference of finite automata.

Takens, F. 1981. Detecting strange attractors in turbulence. *Lecture Notes in Mathematics* 366–381.