Hindawi Publishing Corporation
Journal of Robotics
Volume 2011, Article ID 193146, 16 pages
doi:10.1155/2011/193146

Research Article

Discovering and Characterizing Hidden Variables Using a Novel Neural Network Architecture: LO-Net

Soumi Ray and Tim Oates

Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA

Correspondence should be addressed to Soumi Ray, soumi.ray@gmail.com

Received 1 June 2011; Revised 27 September 2011; Accepted 28 September 2011

Academic Editor: Ivo Bukovsky

Theoretical entities are aspects of the world that cannot be sensed directly but that, nevertheless, are causally relevant. Scientific inquiry has uncovered many such entities, such as black holes and dark matter. We claim that theoretical entities, or hidden variables, 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 discrete time dynamical systems, 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. Two different regularization terms are explored that improve the network's ability to approximate the values of hidden variables, and the performance and capabilities of the network are compared to that of Hidden Markov Models.

1. 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. Just like Gregor Mendel who discovered genes in the middle of the 19th century, you and I cannot see, hear, taste, smell, or touch genes, but we can observe their effects on the world in the color of our friends' eyes or the scent of a rose. That is, genes may be unobservable, but they 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 nondeterminism led Mendel to posit the existence of *a causally efficacious entity of the world that could not be sensed directly*, that is, genes. Such entities are called *theoretical entities*.

No one has ever seen a black hole, yet most physicists believe that they exist because black holes accurately explain a wide array of observational data. We propose to develop algorithms that will allow robots to discover the existence of theoretical entities in much the same way and for many of the same reasons that human scientists do. First, theoretical entities will be posited to explain nondeterminism in action outcomes. For example, a mobile robot in a home environment might discover that sometimes when it is turned on the power required to move forward is 2.5 watts and at other times only 1.5 watts are required, but there is no way to anticipate this prior to actually moving. The robot will posit the existence of a theoretical entity—some aspect of the world that causes this difference but cannot be directly sensed. In this case, we know that the posited entity corresponds to whether the robot starts out on carpet or on a hardwood floor. Second, only those entities that account for multiple nondeterministic observations will be retained. Does knowledge of the fact that the power required to move is 2.5 watts (i.e., the robot is on carpet) make it possible to predict anything else more accurately, such as whether the cleaning function selected by the robot's owner

will be sweep or vacuum? If so, the robot can decide for itself which cleaning function is appropriate simply by moving and measuring power consumption. The goal for the robot, as for the human scientist, is to acquire knowledge that makes it possible to better understand, predict, and control the world around it.

Another example of a domain where finding hidden variables is of great importance is the control of aluminum smelting pots. The United States produces more than 22 billion pounds of aluminum each year. Optimal control of the smelting process is important as suboptimal control not only reduces the yield but also increases energy consumption. But optimal control is very difficult because the smelting process occurs at extremely high temperatures and involves such caustic chemicals that very few sensors are able to survive. This is a case where performance is affected by insufficient sensing. Discovering variables that affect performance, yet are unobservable, can enable better understanding, prediction, and control of the smelting process.

A domain that has motivated much of this work is the discovery of hidden variables where the environment is partially observable with respect to robots. A robot can build a model of the environment to predict future values of state variables based on current and past observations. If hidden variables are present in the environment, then this predictive model will be inaccurate. When the sensor data are insufficient to predict outcomes due to theoretical entities, hidden variables need to be introduced in the model. The main goal of this modeling process is to acquire knowledge that makes it possible to better understand and predict the environment. There are three important tasks to be addressed, each of which is central to this work. The first is to discover the existence of hidden variables; the second is to find the number of hidden variables; and the third is to determine their values.

Predictive models can be developed based on current and past observations in the presence of hidden variables. Hidden variables arise in a wide variety of contexts as an important consideration for explaining observed data. However, most of the previous work in hidden variable (or latent variable or theoretical entity) models have imposed either a simple model or a strong structure on the observed dependencies, be it intervariable dependencies or temporal dependencies. We developed new methodologies that allow for a more data-driven approach to the discovery of hidden variables. This work describes a novel method that has been developed for the discovery of hidden variables using an augmented neural network called the LO-net (Latent and Original network architecture). Almost all of the current hidden variable literature aims to build models that can efficiently learn even when hidden variables are present. Their aim is *not* to discover the hidden variables and quantify them but to maintain or improve predictive performance even when hidden variables might be present.

We compare the LO-net to regular neural networks and also Hidden Markov Models (HMMs). The LO-net always does better than the regular neural network model. The prediction performance using HMMs is comparable to that of the LO-net, but the LO-net provides information about the hidden variables. In HMMs, the hidden states are not informative, hence even if the performance of prediction is good, they do not provide any information about the hidden variables. This work provides a new perspective in finding hidden variables. If hidden variables can be discovered and quantified, then we can have a much better understanding of the overall structure of the system.This can then enable us to find answers to many questions that otherwise could not be accounted for.

Section 2 describes some of the work done on finding hidden variables in different contexts. Section 3 gives a description of the method that we have introduced to find hidden variables, the LO-net. Section 4 lists the different experimental domains that have been used and provides a brief explanation for each of the domains. It describes data obtained from a physical robot data, which is one of our main data sets, and many different dynamical systems. Section 5 presents experimental results using the LO-net on the different experimental domains. Section 6 introduces an extension of the LO-net by adding regularization terms in the network. Section 7 presents experimental results using the regularized LO-net. Section 8 shows the results of comparison of the LO-net architecture with Hidden Markov Models (HMMs). Finally, Section 9 describes the contribution of this work and suggests future directions toward improving these methods.

2. Background

A robot may not be able to observe all of the causally relevant variables in its environment. In the real world, a robot is expected to learn something useful given whatever sensors and effectors are available. Even if the robot's task and the environment change, the hardware typically stays the same. Most commonly, inadequate foresight or sensor technology leads to a gap between what the robot can sense and what it would sense in an ideal world. That is, a robot's environment is almost always *partially observable*. This is particularly problematic in the context of learning because most machine learning algorithms never explicitly consider that there is more to the world than what is provided in their input. For example, decision tree induction algorithms split the data with tests on observed features [1]. Constructive induction algorithms go one step further and build new features out of observed features by combining them in various ways [2]. Kernel methods implicitly project the input data into high-dimensional spaces, some with infinitely many dimensions, but each dimension in these spaces corresponds to a composite feature computed from the inputs [3]. Fahlmans and Lebiere cascade correlation architecture dynamically adds hidden nodes to a neural network, but these nodes do not correspond to hidden features of the environment, they provide a more expressive internal representation based on the inputs [4].

There are many algorithms for learning and reasoning in the face of hidden data, and almost all of them are based on the Expectation Maximization (EM) algorithm [5]. EM is used for parameter estimation when the data required to solve the estimation problem are of the form $X = [Y, Z]$ and Y is observable but Z is not. EM uses a parameterized model

to compute the expected value of the hidden data and then uses the expected value to reestimate model parameters. Therefore, EM requires a priori knowledge about the existence, quantity, and role (in the estimation problem) of hidden data. For example, [6] builds a Bayesian network in which hidden variables are manually added to the network as parents of all observable variables.

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 [7]. 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, [8] showed that dynamical systems can be represented using Predictive State Representations (PSRs), or multistep, action-conditional predictions of future observations, and that every POMDP has an equivalent PSR. PSRs can look at the past and summarize what happened, and they can also look to the future and predict what will happen. A PSR is a vector of tests [9] which stores the predictions for a selected set of action-observation sequences. Unobserved or hidden states can be fully captured by a finite history-based representation called a looping prediction suffix tree (PST) [10]. They focus on cases of POMDPs where the underlying transition and observation functions are deterministic. Again, the goal of this work is not to discover the existence of hidden state, but to behave optimally given knowledge of the existence of hidden state.

Work on hidden states in the context of reinforcement learning has been going on for some time [11]. McCallum et al. proposed a method called *instance-based state identification*, where raw data from previous experiences is stored directly. The simplest such instance-based technique is the *nearest sequence memory*, which is based on k-nearest neighbors. This technique, though simple, improves learning performance. The main disadvantage of this technique is that, though it learns good policies quickly, it does not always learn the optimal policy. There is a relatively new method known as the deep belief nets (DBNs) which are probabilistic generative models that are composed of multiple layers of stochastic, latent variables with no intralayer connections [12]. Gaussian Process Dynamical Models (GPDMs) have also been built to generalize well from small datasets [13].

Latent variables are important in the psychology and social science research. [14] described three definitions of latent variables: *local independence*, *expected value true score*, and *nondeterministic functions of observed variables* and introduced a new notion of latent variables called "*sample generalizations*." Latent variables can be defined *nonformally* or *formally*. Nonformal latent variables can be considered as *hypothetical variables* or unobserved variables as a data reduction device. Hypothetical variables are variables considered imaginary, that is, not existing in the real world. The third nonformal 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.

There is a tremendous body of work on time series analysis, much of it in the neural networks literature [15], with a variety of models aimed at predictive tasks and pattern discovery. That body of literature is too vast to review in any meaningful way here. Suffice it to say that very little of it is devoted to the explicit representation of the number and values of hidden variables in multivariate streaming time series.

In most of the previous work, the aim was to design better models that work with hidden variables. Our aim is to actually discover the hidden variables and quantify them. We describe a method that is developed for the discovery of hidden variables using augmented neural networks.

3. Method

Our neural network architecture is called the LO-net which is, at its heart, a multilayer feed-forward network. Conceptually, it consists of a single O-net (original net) and zero or more L-nets (latent nets). The network and its training are structured in such a way that the single output of each L-net approximates the value of one hidden (latent) variable. The outputs of the L-nets then serve as inputs to the O-net along with the values of the observable variables. The O-net is trained to produce as output the values of the observable variables on the next time step.

Somewhat more formally, let the state of some system at time t be the pair (X_t, Z_t), where $X_t \in \Re^n$ and $Z_t \in \Re^m$. X_t is a vector of n observable values and Z_t is a vector of m unobservable values. Crucially, the state of the system at time $t+1$ depends on both X_t and Z_t so that predictions made using solely X_t, the observable values, will be less accurate than those made using Z_t as well. Standard practice would dictate building a model, f, to predict X_{t+1} based on X_t, that is, $\hat{X}_{t+1} = f(X_t)$, where $\hat{X}_{t+1}$ is a prediction that will be compared against the actual values observed at time $t+1$, that is, X_{t+1}. The model f might be a neural network with n inputs and n outputs and a sufficiently large number of hidden units. Note that the network's hidden layer representation is merely a projection of the observable values into a different (usually higher dimensional) space. Clearly this is key aspect of the performance of neural networks, but the hidden layer representation is not meant to, and typically does not, convey information about unobservable variables, that is, elements Z_t.

One standard way of dealing with hidden state in situations like this is to rely on a history of observable values. A theorem by Takens [16] says 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. Practically, this means that predictions can be improved when the input to the model consists of X_t and some number of additional observation vectors from the recent past. Let $\Phi(X_t, \ldots, X_{t-k+1})$ be a vector of length kn obtained by concatenating X_t, $X_{t-1}, \ldots, X_{t-k+2}, X_{t-k+1}$. In general, a model

predicting X_{t+1} will be more accurate given $\Phi(X_t, \ldots, X_{t-k+1})$ as input for an appropriate value of k than one given simply X_t as input when there is hidden state. While in this case it is reasonable to assume that the hidden layer representation in a neural network trained with $\Phi(X_t, \ldots, X_{t-k+1})$ as inputs will extract information about the hidden variables Z_t, it is impossible to tell from the hidden layer representation how many hidden variables there are (i.e., m) or what their values might be at any given time. The LO-net is explicitly designed to produce exactly that information.

Figure 1 shows the structure of an LO-net with one latent network for which $k = 3$. Note that the latent network is a standard 3-layer feed-forward network that takes as input $\Phi(X_t, X_{t-1}, X_{t-2})$ and produces a single output. The O-net is also a standard 3-layer feed-forward network that takes as input $\Phi(X_t, X_{t-1}, X_{t-2})$ and the output of the L-net. The O-net is trained using backpropagation to predict the value of X_{t+1}. Assuming that $k = 3$ is sufficient, the L-net is initially superfluous. That is, the O-net can ignore the output of the L-net and use the history of values to accurately predict X_{t+1}. Nevertheless, we train the network for some period of time, backpropagating the errors in the usual way through both the O-net and the L-net.

Now that the LO-net as a whole is in a good part of the weight space and can predict X_{t+1} with reasonable accuracy, we change the inputs to the O-net. Let $\mathbf{0}^n$ be a vector of n zeros. We initiate training of the LO-net again, starting with its current weight vector but using $\Phi(X_t, X_{t-1}, \mathbf{0}^n)$ instead of $\Phi(X_t, X_{t-1}, X_{t-2})$ as its input. The L-net is unchanged, and its output is still an input to the O-net. This gives the O-net less information about the hidden variables and causes it to rely more on the output of the L-net. Because the L-net has full access to the history of observables, it is forced to provide information about that history to the O-net through its one output. The most informative and parsimonious way to do that is for the L-net to compute the current value of the hidden variable (assuming $m = 1$) and produce that as output. Note that there is an initial drop in accuracy, but it quickly rebounds as the L-net learns to provide a more informative output. Training continues until the error of the O-net stabilizes, the inputs to the O-net are changed one more time so that they consist of $\Phi(X_t, \mathbf{0}^n, \mathbf{0}^n)$, placing even more pressure on the L-net to provide information about the value of the hidden variable, and training continues until the error of the O-net stabilizes.

To estimate the number of hidden variables, we run this procedure with a sequence of LO-nets starting with no latent networks. If adding an additional latent network significantly improves prediction accuracy, we have evidence for a hidden variable. We keep adding latent networks until doing so has no significant impact on the accuracy of the O-net at predicting X_{t+1}. Empirically, we have strong evidence that the number of latent networks in the previous LO-net corresponds to the number of hidden variables and their outputs are approximations (up to a scaling factor) of the values of the hidden variables. Note that in contrast to a standard neural network (and most other kinds of models one might use in this case), the LO-net makes explicit information about hidden variables.

Note that this method makes no assumptions about the numbers of observable or unobservable variables and will work with histories of arbitrary size. Figures 2(a) and 2(b) show the one- and two-LO-net architectures, respectively. Note that in the latter the outputs of both L-nets act as inputs to the O-net and their values are presented in parallel.

The L-net and the O-net have the same network structure. The neural network architecture has been implemented in Matlab, using Matlab's Neural Network Toolbox. The Neural Network Toolbox provides a flexible network object type that allows many kinds of networks to be created and then used with other functions in the toolbox. This flexibility is possible because the networks have an object-oriented representation, allowing a user to define various architectures and assign different algorithms to those architectures. To create a custom network, we start with an empty network (obtained with the network function) and set its properties as desired. The network object consists of many properties that can be set to specify the structure and behavior of the network.

In our implementation, the network has each of its layers' weights and biases initialized with the Nguyen-Widrow layer initialization method [17]. The Nguyen-Widrow method generates initial weight and bias values for each 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. 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 learning rate (lr) chosen is 0.01. The momentum constant used was 0.9. 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 that increased the performance is not made. A transfer function is used to calculate the ith layer's output given the layer's net input during simulation and training. Backpropagation is used to calculate the derivatives of performance (perf) with respect to the weight and bias variables X, where performance is measured according to the mean squared error. Each variable is adjusted according to gradient descent with momentum given by

$$dX = \text{mc} * dX\text{prev} + \text{lr} * (1 - \text{mc}) * \frac{d\text{perf}}{dX}, \tag{1}$$

where dXprev is the previous change to the weight or bias. Gradient descent with momentum depends on two training parameters. 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. The hyperbolic tangent sigmoid transfer function takes the net inputs and returns a value squashed into $[-1, 1]$.

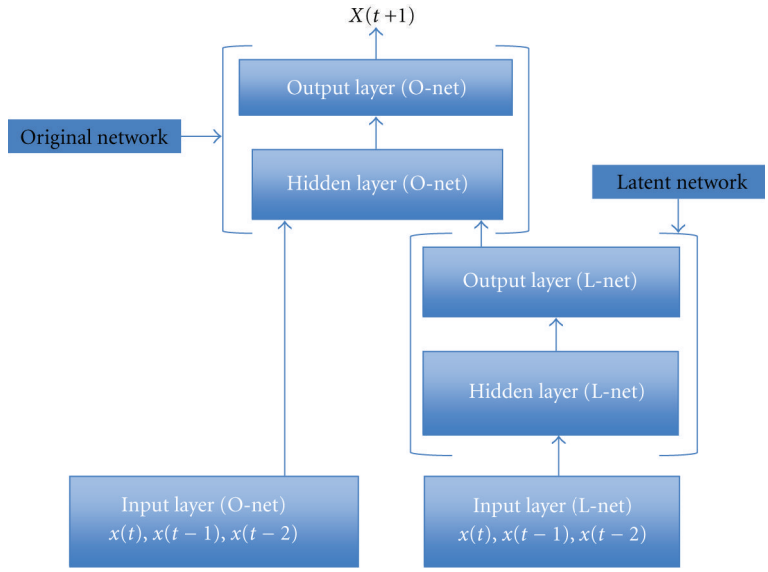
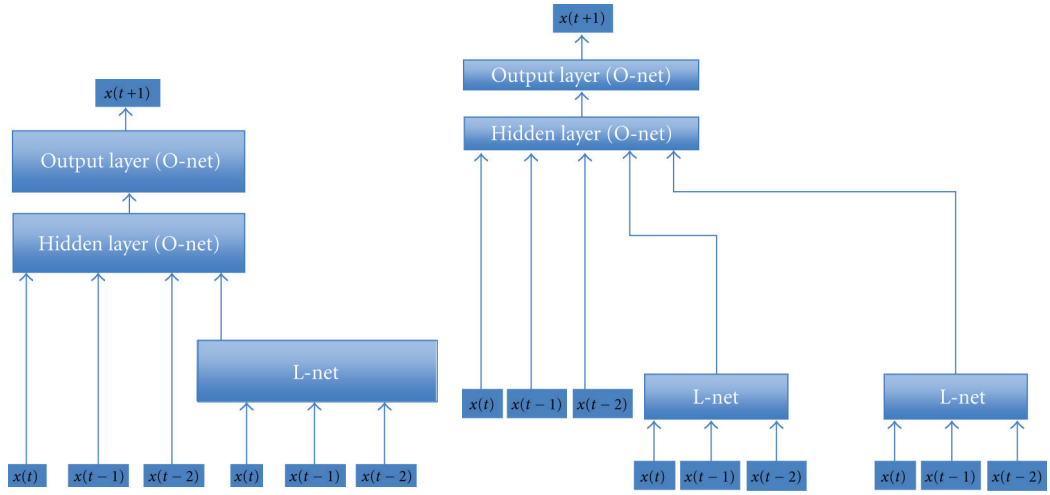


Figure 1: A generic LO-net with one latent network.



(a) An LO-net with one L-net

(b) An LO-net with two L-nets

Figure 2: Example LO-net architectures.

4. Test Domains

This section describes the various domains from which data were drawn to test the LO-net approach. In all cases, the goal is to determine the number of hidden variables and to approximate, as accurately as possible, their values. In practice, the trained LO-net would be used in two qualitatively different ways. The first is by humans to better understand the domain. Knowing, for example, that there are three hidden variables that affect the price of a financial security or the blood pressure of an intensive care unit patient, and being able to see how they change over time would be extremely valuable for domain experts. The second way in which the results would be used is by machines for making more accurate predictions. The output of the O-net is more accurate by virtue of having the L-nets, and the values produced by the L-nets can be used as inputs to other models constructed for other purposes in the underlying domain.

4.1. Robot Data. We take a robot mounted camera taking pictures of 2D objects. Consider the coordinate frame shown in Figure 3, center of projection at the origin and an image plane orthogonal to the z axis. The robot's vision system extracts the following information for a given box:

s_t: the size of the object in the image plane,

x_y: the x coordinate of the centroid of the object in the image plane,

y_t: the y coordinate of the centroid of the object in the image plane.

Each object has an objective size S_t and objective location (X_t, Y_t, Z_t), relative to the origin of the coordinate frame.

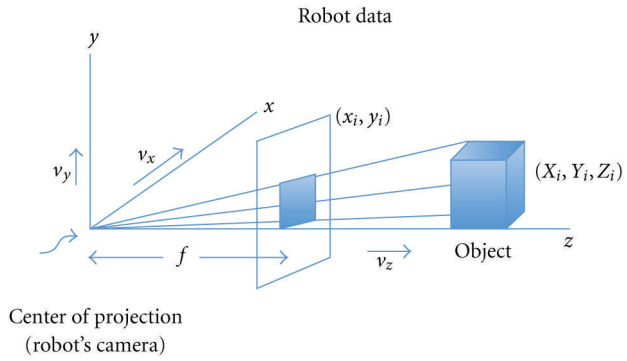


Figure 3: Perspective projection of the centroid of an object in 3D space onto the robot's image plane.

Assume a simple case where the robot is moving along the z-axis with a constant velocity $v_z = 1$. The quantities required to predict the next value of x_{t+1} and y_{t+1} are observable except Z, the distance of the robot from the box.

$$x_{t+1} = x_t + \frac{v_z x_t}{z_t},$$
$$y_{t+1} = y_t + \frac{v_z y_t}{z_t}. \tag{2}$$

The perceived size of an object s_t depends on the objective size S and the distance Z of the object as follows:

$$s_t = \frac{S_t}{Z_t^2}. \tag{3}$$

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 and Z are not observable, so s cannot be directly estimated.

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 through 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 several targets. The targets are brightly colored boxes, easily distinguishable from the surrounding environment by standard image processing software. The surveyor would approach a target from different angles, keeping it in view the entire time for some trials, and occasionally facing in different directions for others. 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.

4.2. Lorenz Attractor. The Lorenz attractor [18] is a dynamical system which has three variables x, y, and z. It is defined by a system of three equations that give the first derivative of each of the three variables

$$\frac{dx}{dt} = \sigma(y - x),$$
$$\frac{dy}{dt} = x(\rho - z) - y, \tag{4}$$
$$\frac{dz}{dt} = xy - \beta z.$$

We can test our methods using the Lorenz attractor, or any other dynamical system, by generating data for x, y, and z over time and then hiding one or two of the variables.

4.3. Character Trajectories Dataset. We also used a practical data set from the UCI machine learning repository [19], the character trajectories data set. The data consists of 2858 character samples. We randomly picked one character sample from the data set to run the experiments. The data stored for each character was the x and y coordinates and the pen tip force during writing a character and were captured at 200 Hz. The pen tip force will depend on both the x and y coordinates.

4.4. A Dynamical System with Four Variables. Next we generated a dynamical system with four variables, x, y, z, and h. This dynamical system is defined by the following set of equations:

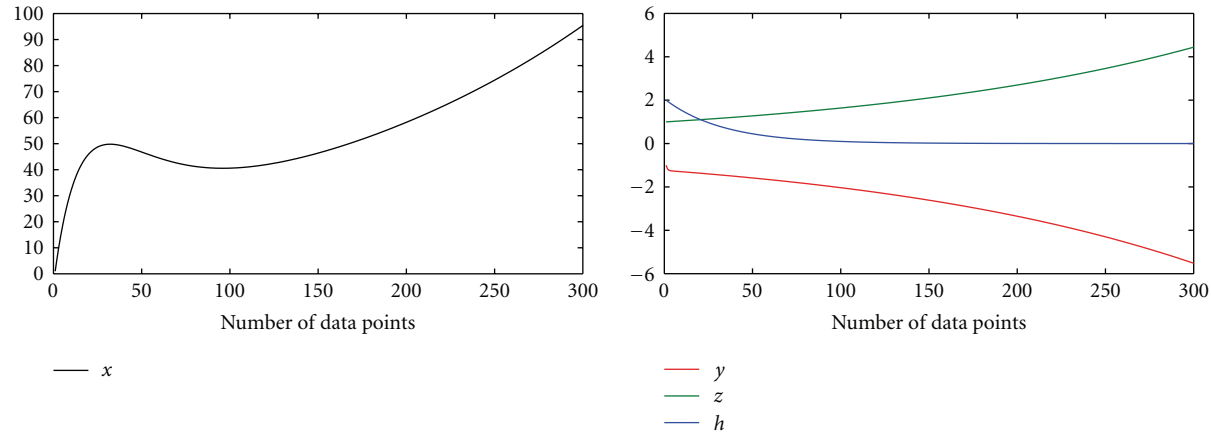
$$x_{t+1} = ay_t + cx_t + z_t + h_t^2,$$
$$y_{t+1} = ay_t - z_t,$$
$$z_{t+1} = bz_t, \tag{5}$$
$$h_{t+1} = ch_t,$$

where $a = 0.2$, $b = 1.005$, and $c = 0.97$. The initial values of the variables are $x_0 = 1$, $y_0 = -1$, $z_0 = 1$, and $h_0 = 2$. Figures 4(a) and 4(b) plot the values of (x_t, y_t, z_t, h_t) over time.

Note that the variable x depends on all four variables x, y, z, and h. This dynamical system will be referred henceforth as the *four-variable dynamical system* (FVDS).

5. Experimental Results Using the Unregularized LO-Net

5.1. Simulated Robot Data. The first set of experiments are performed with simulated robot data. A robot collects data by moving towards a box with a constant velocity. It records the x and y coordinates of the centroid of the box and the size of the box in its vision. The network is trained for 400 epochs since around that time the mean squared error (MSE) converges to a very low value. The dataset is divided into training and test datasets. The MSE is calculated over the test dataset. Figure 8(a) plots the MSE for predicting the next value of x. The solid line shows the performance when the current value of x (x_t) is input and 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



(a) Variable x

(b) Variables y, z, and h

FIGURE 4: Sample values from the four-variable dynamical system.

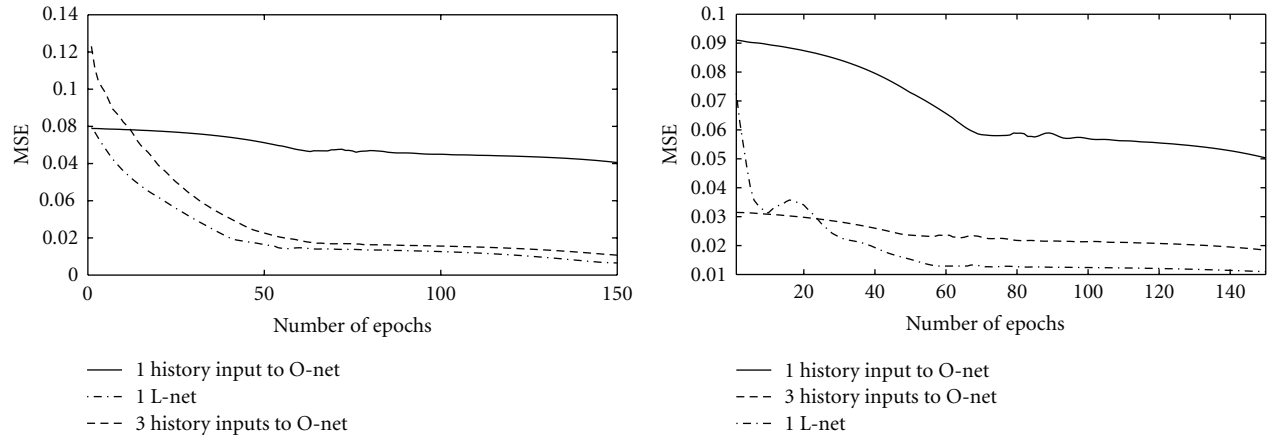
two x values (x_t, x_{t-1}, x_{t-2}) are input to the original network and the next value of x x_{t+1} is predicted. The dash-dot line and the dotted lines show the performance with one and two latent networks, respectively. Initially for the first 100 epochs, the current and the previous two x values (x_t, x_{t-1}, x_{t-2}) are input 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. 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 figures plot the MSE versus the number of epochs. The plots show the MSE of the last 150 epochs where the input to the original net is the current value of the variable and the output from the latent net. The x-axis plots the number of iterations and the y-axis plots the mean square error (MSE) in the following figures.

Figures 5(a) and 5(b) plot the performance curve for predicting x_{t+1} and y_{t+1}, respectively. In both cases, only z is hidden. It is observed that the maximum performance improvement is achieved using one-LO-net architecture in both cases. Figure 6 plots the performance for predicting the size s of the box when there is more than one box in the robot's field of vision. There are two hidden variables: z, the distance of the robot from the box, and S, the actual size of the box. The performance is best in this case with the two-LO-net architecture. The performance decreases when a third latent network is added. Figure 7 shows the outputs from the L-nets in the two-LO-net architecture for predicting the size s of the box.

5.2. Real Robot Data. The next set of experiments are performed with real robot data. The experimental setup is described above. It is the exact same scenario as for the simulated robot data. From Figure 8(a), the performance of the network with three history values for predicting x_{t+1} is better than the performance with just the current value. The one-LO-net architecture performs better than the two-LO-net architecture. 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 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 net 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 when more learning history values are dropped, the input from the latent network becomes more important. We propose that the backpropagation algorithm updates the weights of the latent network so as to approximate the hidden variable. Similar results can be seen in the case of predicting y_{t+1}. The two-LO-net architecture performs best when predicting the size of the box as seen in Figure 8(b). The size of a box depends on the actual size of the box and the distance of the box from the robot. Hence for predicting the next value of the size perceived by the robot, the two hidden variables are the actual size of the box and the distance of the robot from the box. 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 suggest the existence of hidden variables, but it also gives an estimate of the number of hidden variables. For example, in this case where x depends only on x and Z where Z was hidden, one-LO-net architecture gave the maximum performance improvement. While predicting future values of s which depends on two hidden variables S and Z, the two-LO-net architecture performs the best.

5.3. Lorenz Attractor. For the Lorenz attractor, the performance of the network with three history values for predicting x_{t+1} is better than the performance with just the current value. The performance of the one-LO-net architecture is the best in this case as seen in Figure 9(a). It performs better than the two-LO-net architecture. The variable x in the Lorenz



(a) MSE when predicting x_{t+1}

(b) MSE when predicting y_{t+1}

Figure 5: Simulated robot data.

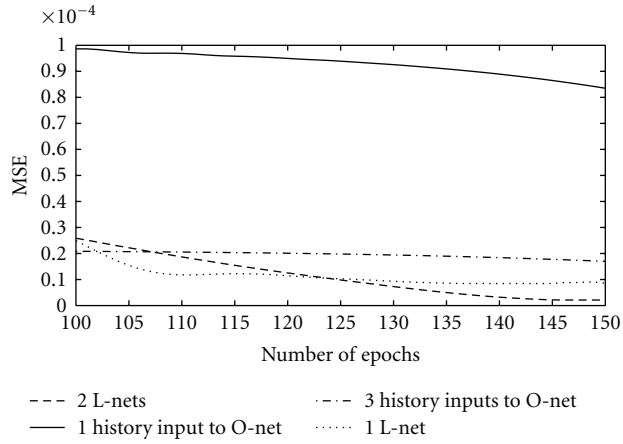
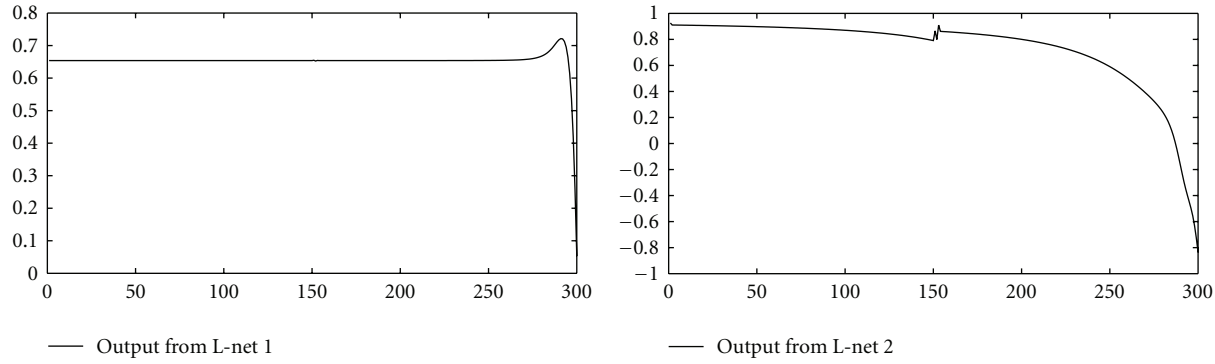


Figure 6: Simulated robot data: MSE when predicting s_{t+1}.

attractor depends only on variables x and y. While trying to predict the next value of x with just the previous value of x, variable y is missing on which x is dependent. The output from the latent network in the LO-net architecture provides input to the original network that improves its performance. Figure 9(b) shows the plot of the MSE for predicting variable y. Variable y depends on all the three variables x, y, and z. Here the performance is best in the case of the two-LO-net architecture. One latent network does improve the performance from that of using just the original network with history inputs. The two latent networks in this case approximate the two hidden variables x and z.

5.4. Character Trajectory Dataset. The LO-net was also tested on a practical data set from the UCI machine learning repository [19], the character trajectories data set. The network predicts the next value of the pen tip force using the current and the history values of the pen tip force. Since the pen tip force depends on both the x and y coordinates these are the two hidden variables. Here the network is trained for 300 epochs and the plots show the MSE of the last 100 epochs. Figure 10 shows that the two-LO-net architecture gives the maximum performance improvement for predicting the next value of the pen tip force.

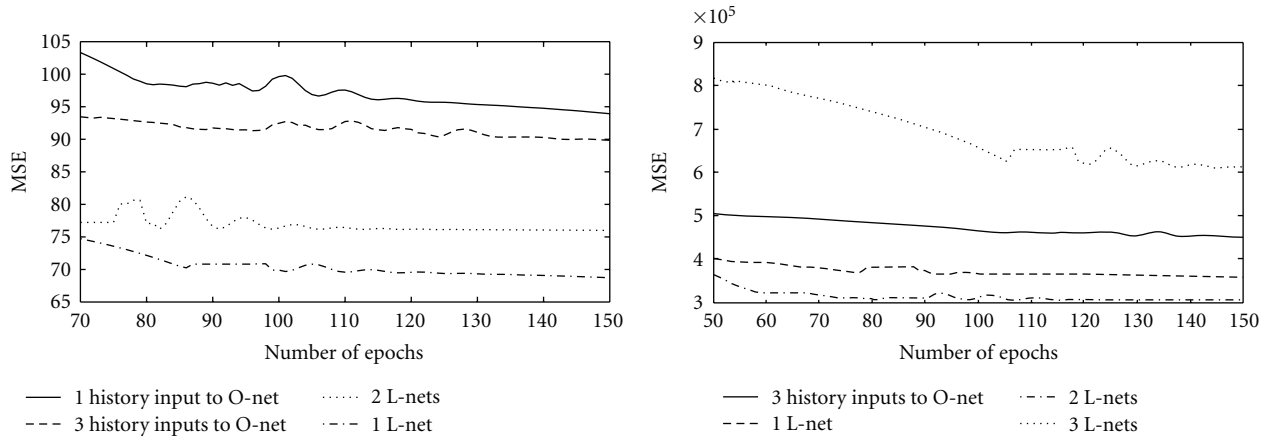
5.5. Four-Variable Dynamical System (FVDS). Figure 11 shows the performance curve for predicting x_{t+1}. The value of x_{t+1} depends on the variables x_t, y_t, z_t, and h_t. The O-net and the LO-net are provided with only the current and history values of x. Hence, for predicting x_{t+1}, there are three hidden variables present in the system. The maximum performance improvement for predicting x_{t+1} is achieved by the LO-net architecture with three L-nets as seen in Figure 11. When a fourth latent network is added, the performance decreases. These results show that the LO-net architecture works well even in the presence of larger numbers of hidden variables. It can detect the true number of hidden variables, and scaling this architecture up as the number of hidden variables increases works well. The correlations between the values of the hidden variables and the outputs from the L-nets are computed. In the one-LO-net architecture, the output from L-net approximates the value of the output from the O-net. As a result, the performance using the one LO-net architecture improves over the performance with three history values as input to the O-net, as seen in Figure 12. Figures 13, 14, and 15 compare the nature of the outputs from the L-nets and the hidden variables in the three-LO-net architecture. The correlation between the output from L-net 2 and the hidden variable y is 0.8964. Correlation between output form L-net 1 and hidden variable z is 0.8499. The correlation between the output from L-net 3 and the hidden variable h is 0.6808. We have developed a neural network architecture that 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.



(a) Output from L-net 1 for predicting s_{t+1} in 2-LO-net architecture (z and S are the hidden variables here)

(b) Output from L-net 2 for predicting z_{t+1} in 2-Lo-net architecture (z and S are hidden variables here)

Figure 7: Simulated robot data: outputs from the L-nets.



(a) MSE when predicting x_{t+1}

(b) MSE when predicting the size s_{t+1} of the box with two boxes in the robot's field of vision

Figure 8: Real robot data.

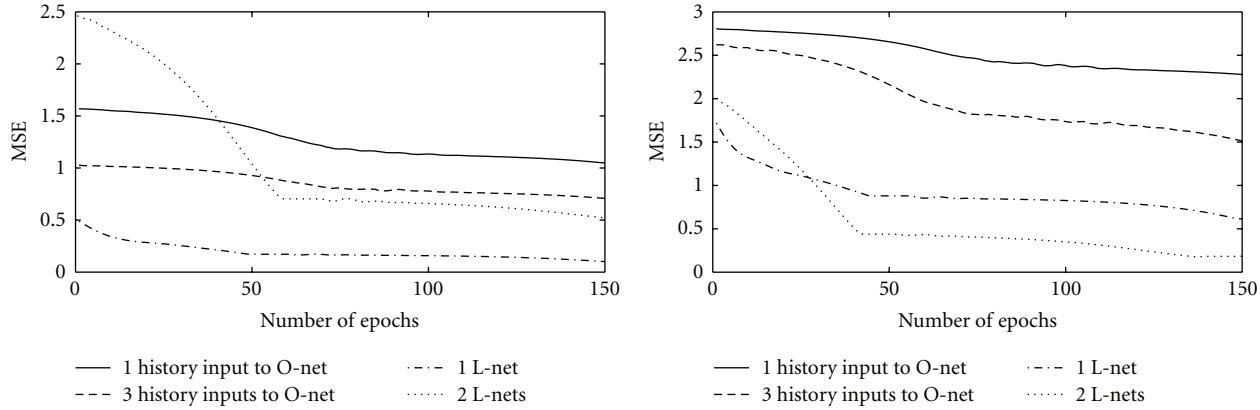
6. Regularization of the LO-Net Using Two Different Penalty Terms

The latent network sometimes tries to approximate the predicted output from the original network in the LO-net architecture. When we are trying to predict x_{t+1} with history values of x as input to the LO-net, the L-net can sometimes try to approximate the predicted value x_{t+1} instead of the hidden variable, that is y in this case. Since the latent network is not provided with any example outputs, the only way it learns is from the errors back-propagated to it. In many cases, the predicted variable and the hidden variable are highly correlated; then the latent variable might just approximate the predicted variable. Even when the predicted variable and the latent variable are not correlated, the overall network may "cheat" by forcing the latent network to do all of the work. We introduce a penalty term which is added to the error for the latent network to prevent this situation. The penalty term is formulated so that the latent network does not learn the output of the original network. The penalty term is applied to the output of the L-nets, and the gradient descent calculation is modified to include both the weight changes back-propagated from the output of the O-net and the weight changes suggested by the penalty term. That is, there is now pressure inside the L-nets to change the weights so as to minimize the penalty term in addition to the pressure to change the weights to improve the accuracy of the LO-net as a whole. These pressures are balance by selecting the value for a single parameter.

6.1. Distance Penalty P1. The first penalty term is given by

$$P = w * e^{-(O-L)^2}, \tag{6}$$

where O is the output from the original network, that is, the output of the whole network and L is the output of the latent network, and w is the weight of the penalty term which lies between zero and one [20]. Whenever the output from the original network is close to the output from the latent network, the penalty term is high, otherwise it is low. If the value of w is high, then the penalty term plays an important role in the learning of the latent network. The output from the latent network in this case will be



(a) MSE when predicting variable x_{t+1}

(b) MSE when predicting variable y_{t+1}

Figure 9: Lorenz attractor.

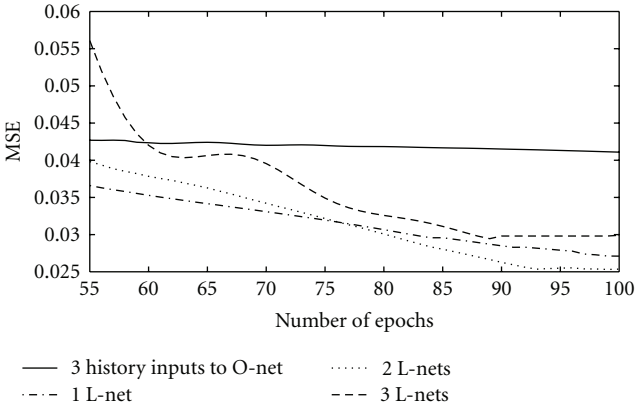


Figure 10: Performance curve for the pen tip force (character trajectories data set from the UCI machine learning repository).

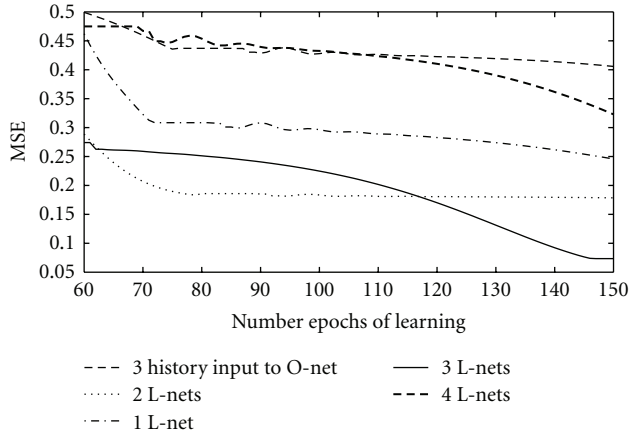


Figure 11: FVDS: Performance curve for predicting x_{t+1} with O-net (inputs are three history values x_t, x_{t-1}, and x_{t-2}) and LO-net architecture with one, two, three, and four L-nets.

significantly less correlated to the output from the original network. If the predicted variable and the hidden variable are highly correlated, then a high value of w will result in the latent net not learning the hidden variable because it will get a high penalty on being close to the predicted output and will deviate from that. Deviating from learning the predicted output will also move the network from learning the hidden variable since the hidden variable might be very close to the predicted output. Hence choosing an appropriate value of w for maximum performance improvement is crucial.

In a case where the two-LO-net architecture gives the maximum performance improvement, there are two different penalty terms for the two latent networks. For the first latent network, the penalty is same as before that is,

$$P = w * e^{-(O-L1)^2}, \tag{7}$$

where again O is the output from the original network and $L1$ is the output from the first latent network. The penalty term for the second latent network is

$$P = w * e^{-(L1-L2)^2}, \tag{8}$$

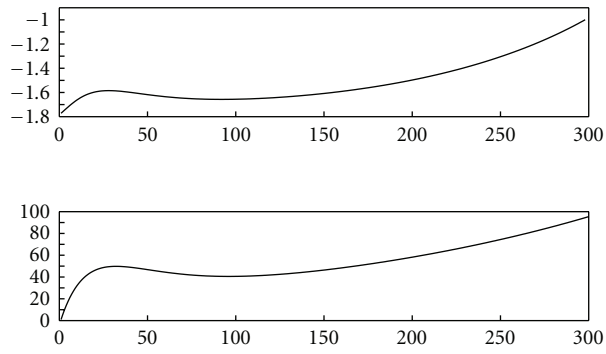


Figure 12: FVDS (one-LO-net architecture): comparing output from L-net 1 and variable x.

where again $L1$ is the output from the first latent network and $L2$ is the output from the second latent network. We do not want the second latent network to learn the output from the first latent.

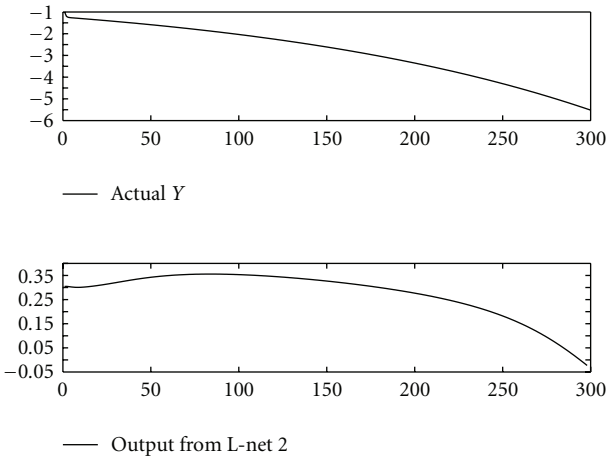


Figure 13: FVDS (three-LO-net architecture): comparing output from L-net two and hidden variable y, correlation is 0.8964.

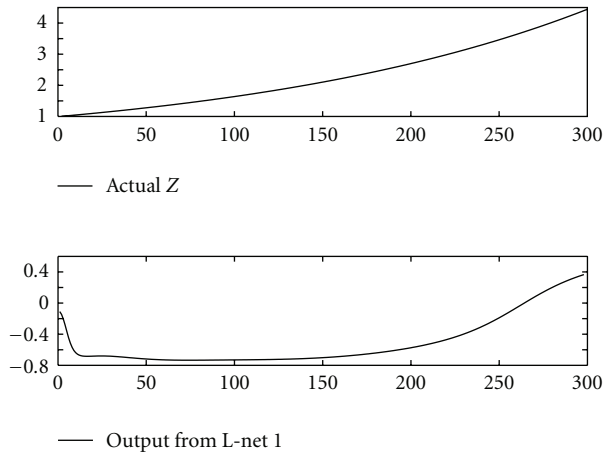


Figure 14: FVDS (three-LO-net architecture): comparing output from L-net one and hidden variable z, correlation is 0.8964.

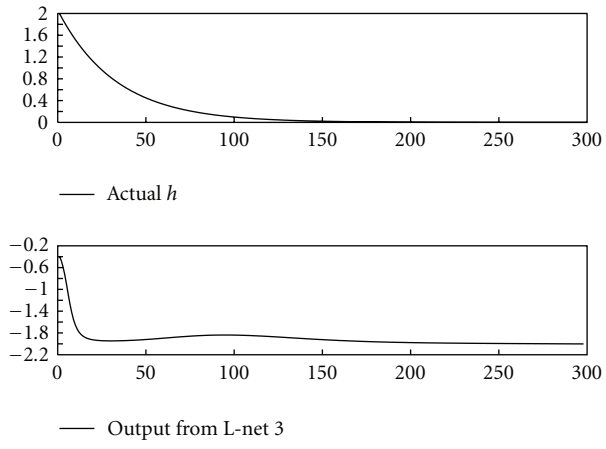


Figure 15: FVDS (three-LO-net architecture): comparing output from L-net 3 and hidden variable h, correlation is 0.6808.

6.2. Decorrelation Penalty P2. In some cases, the output from the latent network has to be decorrelated from the output of the original network. Decorrelation is a process that is used to reduce the autocorrelation within a dataset or the cross-correlation between two data sets while preserving the other aspects of the data. The extent of decorrelation depends on the weight of a decorrelation penalty. A new penalty term $P2$ is introduced which is the square of the covariance between the output of the O-net and L-net. The idea of this penalty term was adopted from [21]. The penalty term is

$$P2 = (\mathrm{Cov}_t(O, L))^2, \tag{9}$$

where O is the output from the O-net and L is the L-net. This can be written as

$$P2 = [\mathrm{mean}(O,\ L) - \mathrm{mean}(O)\ \times\ \mathrm{mean}(L)]^2. \tag{10}$$

The neural network toolbox in Matlab provides a method, calcgrad.m, which computes the gradient of the error with respect to the input weights, layer weights, and bias weights. We adapted the code in this file to incorporate the penalty term. The gradients of the penalty term with respect to the layer weights of the hidden layer in the latent network are computed and added it to the gradients of the error. The derivative of the penalty term $P2$ with respect to the weights of the hidden layer can be written as

$$\frac{dP2}{dw} = 2 \times P2 \times \frac{dP2}{dL} \times \frac{dL}{dw}, \tag{11}$$

where w is the weight vector of the hidden layer of the latent network. The derivative of L with respect to w is the output from the hidden nodes, $I_{h\times N}$, where h is the number of neurons in the hidden layer and N is the number of instances of the training set, that is

$$\frac{dL}{dw} = I. \tag{12}$$

Note that O and L are vectors of size N.

Next the derivative of $P2$ with respect to L (the output of the latent network) is computed. We know that

$$P2 = [\mathrm{mean}(O,\ L) - \mathrm{mean}(O)\ \times\ \mathrm{mean}(L)]^2. \tag{13}$$

This can be written as

$$P2 = \frac{O \cdot L}{N} - \left(\frac{O \cdot A}{N}\right) * \left(\frac{L \cdot A}{N}\right), \tag{14}$$

where A is a vector of ones of size N.

$$\begin{aligned}\frac{dP2}{dL} &= \frac{d(O \cdot L/N - (A \cdot O/N) \times (A \cdot L/N))}{dL} \\ &= \left(L \cdot \frac{dO}{dL} + O \cdot \frac{dL}{dL}\right)/N \\ &\quad - \left(\left(A \cdot \frac{dO}{dL}\right)/N \times \left(\frac{A \cdot L}{N}\right)\right) \\ &\quad + \left(\left(A \cdot \frac{dL}{dL}\right)/N \times \left(\frac{A \cdot O}{N}\right)\right),\end{aligned} \tag{15}$$

where dO/dL and dL/dL are $h \times N$ matrices computed numerically. Finally the gradient of the penalty with respect to the weights is

$$\frac{dP2}{dw} = 2 \times P2 \times \frac{dP2}{dL} \times L, \tag{16}$$

which is a vector of size h. These gradients are added to the gradient of the errors with respect to the weights of the latent network.

7. Experimental Results Using the Regularized LO-Net

7.1. Simulated Robot Data. The results of using the regularized LO-net on the simulated robot data are shown in this section. The performance of predicting x is best with the one LO-net architecture using the first penalty term $P1$ with weight $w = 0.0001$ (Figure 16(a)). The performance of predicting y is also best with the one LO-net architecture using the first penalty term $P1$ with weight $w = 0.0001$ (Figure 16(b)).

7.2. Lorenz Attractor. Regularization of the latent network improves the approximation of the hidden variables in the Lorenz Attractor domain. In Figure 17(a), the solid line shows the performance of predicting x with three history values of x. The dash-dot line shows the performance with the one LO-net architecture. The dash line shows the performance with the one LO-net architecture using the first penalty term (squared error $P1$), and the dotted line shows the performance with the second penalty term (covariance $P2$). It is observed from Figure 17(a) that the regularization with the second penalty terms performs the best in this case. The next value of y in the Lorenz attractor depends on both x and z, hence there are two hidden variables. As seen before, in this case, the two-LO-net architecture performs the best. The maximum performance improvement is achieved with the second penalty term $P2$, as seen in Figure 17(b). Similar results are observed when predicting z. The decorrelation penalty term performs better than the distance penalty term for cases where the observed variables are periodic.

7.3. Real Robot Data. Next the effect of adding a penalty term in the latent network to that of using no penalty is compared in the real robot data domain. All of the results shown are the best of ten iterations of the same experiment. Both the x and y coordinates of the box observed by the robot depend on the hidden variable Z, which is the distance of the robot from the box. Therefore, the maximum performance improvement is achieved using the one LO-net architecture, as explained before. Figures 18(a) and 18(b) show the performance of adding a penalty term to the error back-propagated to the latent network for predicting the x and the y coordinates. Similar results were seen in the domain when predicting the y coordinate with regularization. For predicting the next value of x, x_{t+1}, the maximum performance improvement is achieved for $w = 0.05$. For predicting y_{t+1}, the maximum improvement in performance is achieved for $w = 0.5$. The robot in the real world randomly starts from a point in the coordinate space and moves towards the object. In our set of experiments, the values of variable x are higher than the values of variable y. Therefore, the outputs from the O-net are larger for x than for y. It is observed that the outputs from the L-nets are small in both cases, which makes the difference $(O - L)$ larger for x compared to y. Hence the maximum improvement in performance is achieved with a smaller value of weight in the case of x than in the case of y. It is seen that the value of w plays an important role in the amount of performance improvement.

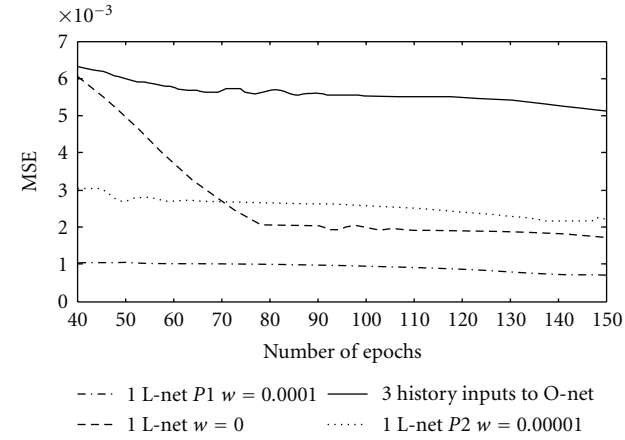
Figure 19 shows the effect of adding a penalty term for predicting the next value of size, s_{t+1}. The size of the box observed by the robot depends on two hidden variables: S, the actual size of the box and Z, the distance of the robot from the box. In this case, the two-LO-net architecture gives the maximum performance improvement. The maximum performance improvement is achieved for $w = 0.005$.

8. Comparison of the LO-Net Architecture with Hidden Markov Models (HMMs)

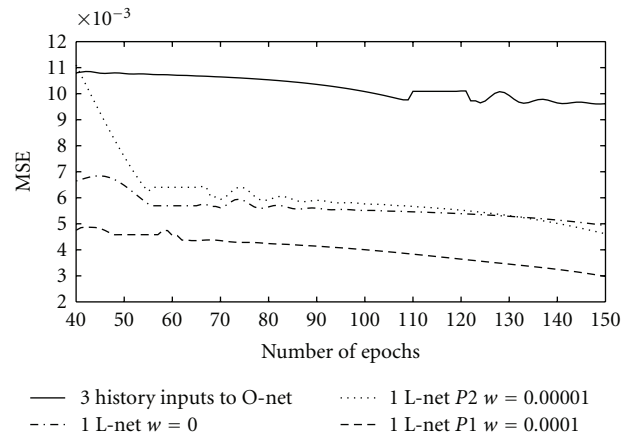
Hidden Markov Models (HMMs) are Markov processes with unobserved or hidden states [22]. In HMMs, the states are not directly observable, but the outputs, which are dependent on the states are observable. HMMs have a wide variety of applications in speech and handwriting recognition, music scores, and other time series domains. We have used ergodic continuous HMMs, in which the outputs given the hidden states are modeled using Gaussian distributions. Maximum likelihood parameter estimation is performed using the Baum-Welch algorithm. The Verterbi path (the most probable sequence of the hidden states) is computed, and the predicted output at each time step is computed as the estimated mean of the Gaussian distribution conditional on the most probable hidden state at that time step. Tables 1 and 2 show the MSEs for predicting x and z variables of the Lorenz attractor data using HMMs and LO-net. The next value of variable x depends on the current values of the variables x and y. There is one hidden variable for predicting x, namely, y. The next value of variable z depends on the current values of x, y, and z. Hence there are two hidden variables for predicting z, namely, x and y. The MSEs for predicting x using both the models are very similar again as seen in Table 1. The MSE for predicting z is lower using the LO-net architecture than the HMM model shown in Table 2.

Tables 3 and 4 show the MSEs of predicting the x- and y-coordinates in the simulated robot data using HMMs with the number of states varying from 100 to 250, as well as the LO-net. The input to the networks is a vector of 300 timesteps of the values of the observed variable. Recall that both x and y depend on one hidden variable, the distance of the robot from the object z. The MSEs for prediction are very similar using both the HMM model and the LO-net architecture.

Table 5 shows the MSEs for predicting the pen tip force from the Character Trajectory Dataset again using HMMs and LO-net. In this case, LO-net performs better than the

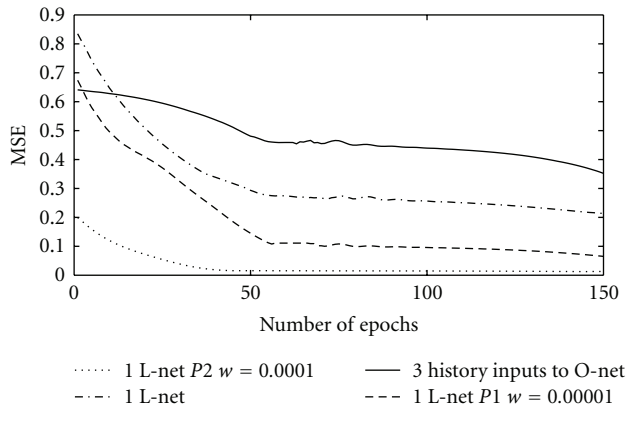


(a) Performance curve for predicting x_{t+1} (net with regularization)

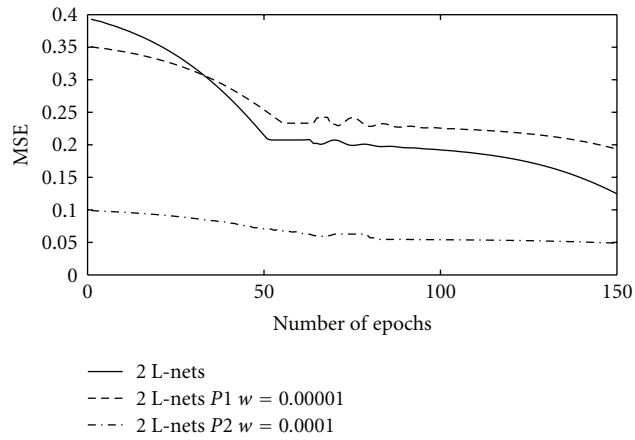


(b) Performance curve for predicting y_{t+1} (net with regularization)

Figure 16: Simulated robot data.

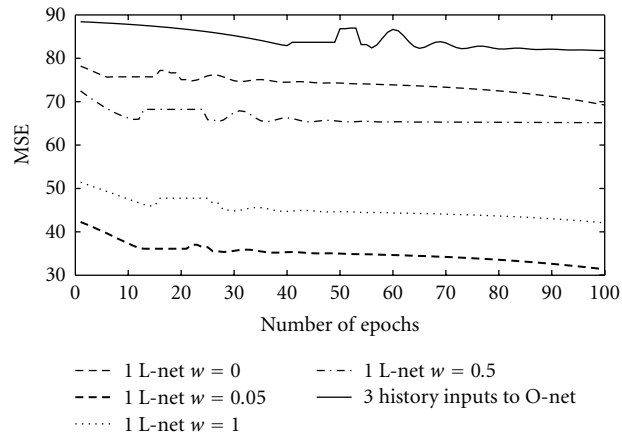


(a) Performance curve for predicting variable x_{t+1} (net with regularization)

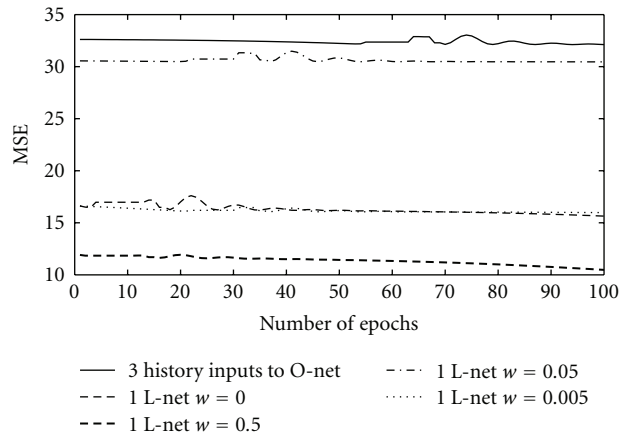


(b) Performance curve for predicting variable y_{t+1} (net with regularization)

Figure 17: Lorenz attractor.



(a) Performance curve for predicting x_{t+1} with different values of the weight (w) for the distance penalty term ($P1$)



(b) Performance curve for predicting y_{t+1} with different values of the weight (w) for the distance penalty term ($P1$)

Figure 18: Real robot data.

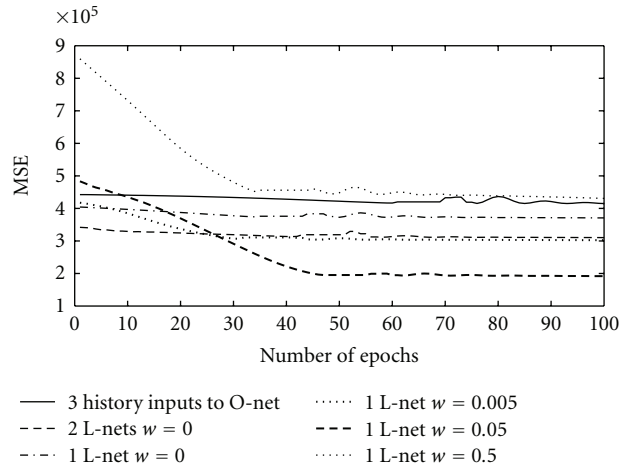


Figure 19: Real robot data: performance curve for predicting s_{t+1} with different values of the weight (w) for the penalty term (distance penalty $P1$).

Table 1: Lorenz attractor (predicting x_{t+1}): mean squared errors (MSEs).

HMM		LO-net
No. of states	MSEs	MSE
150	0.0383	0.0108
200	0.0275	
250	0.017	

Table 2: Lorenz attractor (predicting z_{t+1}): mean squared errors (MSEs).

HMM		LO-net
No. of states	MSEs	MSE
150	0.0739	0.058
200	0.071	
250	0.0701	

Table 3: Simulated robot data (predicting x_{t+1}): mean squared errors (MSEs).

HMM		LO-net
No. of states	MSEs	MSE
100	0.004	0.0008
150	0.0036	
200	0.0027	
250	0.00086	

HMMs. The pen tip force depends on two hidden variables the x and the y coordinates of the pen tip on the paper.

The performances using the HMM model and the LO-net architecture are comparable in the presence of one hidden variable. It is observed that the LO-net architecture performs better than the HMM model when two hidden variables are present. This might be because when there is one hidden variable the hidden states correspond to the one hidden variable. When there are two hidden variables, more hidden states are needed to capture the nature of the two hidden variables. The HMM model does not scale well as the number of hidden variables increases.

Table 4: Simulated robot data (predicting y_{t+1}): mean squared errors (MSEs).

HMM		LO-net
No. of states	MSEs	MSE
150	0.071	0.0045
200	0.0099	
250	0.0076	

Table 5: Character trajectories dataset (predicting the pen tip force): mean squared errors (MSEs).

HMM		LO-net
No. of states	MSEs	MSE
50	0.0157	0.0067
100	0.0131	

The HMM model does improve prediction in the presence of hidden variables, but it does not provide any information *about* the hidden variables. Adding hidden states to the HMM can improve its predictive performance, but the number of hidden states in the model is not a reflection of the number of hidden variables in the system. Our LO-net architecture, on the other hand, is successful not only at discovering the number of hidden variables but also in providing estimates of the values of the hidden variables.

9. Conclusion and Future Work

We presented a novel neural network architecture, the LO-net, that was able to address three basic tasks related to hidden variables: (i) discovery, (ii) cardinality, and (iii) estimation. The LO-net is a combination of two networks, the O-net and L-nets, where the outputs from the L-nets are inputs to the O-net. The network is trained using gradient descent backpropagation. History values are provided as inputs to both the networks in the LO-net. As learning progresses, history values are dropped from the O-net. At the end of learning, the L-nets approximate the values of the hidden variables. Experiments showed the utility of the proposed approach using 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. The LO-net was also tested on a variety of nonlinear dynamical systems and simulated data.

In the real robot data, which is a domain that has motivated much of this work, we observed that the performance for predicting x_{t+1} and y_{t+1}, which are the coordinates of an object in the robot's visual field, improved by adding an L-net to the O-net. In the robot's field of vision, the distance of the robot from the object is hidden and the x and the y coordinates depend on this hidden variable. We also found that the output from the L-net approximated the value of the hidden variable. The output from L-net

does not provide the exact value of the hidden variable but its output approximates the effect of the hidden variable. Outputs from the L-nets were shown and compared to the values of the hidden variables in the presence of one or two hidden variables. The correlations between the outputs from the L-nets and the values of the hidden variables were computed; in most cases the correlations were high (in the range of 0.4 to 0.9). However, in some cases the outputs from the L-nets are more correlated with the output from the O-net than that with the hidden variables. That is, the L-nets were approximating the output from the O-net and not the values of the hidden variables. This led to regularize the L-nets to prevent the L-net from mimicking the output from the O-net.

A penalty term was added to the error back-propagated to the L-net. Experiments were performed using two different forms of the penalty term. The first penalty term, $P1$, is the square of the difference between the outputs from the O- and L-nets, which provides a high penalty if the O- and L-net outputs become similar. This penalty term is successful in pushing the outputs from the L-nets away from the O-net in some domains. However, $P1$ cannot always reduce the correlation between the outputs from the L-nets and O-net. A second penalty term, the decorrelation penalty, $P2$, is introduced to try to decorrelate the two outputs. This additional penalty is able to improve performance of the LO-net in some domains. The performance of the regularized LO-net is compared to that of the vanilla (unregularized) LO-net across many different dynamical systems as well as our robot data.

We also compared the performance of the LO-net with HMMs and found them to be comparable when only one hidden variable is present. The LO-net performs better than HMMs when two hidden variables are present. A disadvantage of HMMs compared to the LO-net is that HMMs do not scale well with the number of observations or the number of hidden variables and do not provide estimation and cardinality detection of the hidden variables unlike the LO-net. The LO-net has the advantage of providing insights into the nature of the system being investigated through its ability to detect and estimate hidden variables.

While our results are extremely encouraging, there are clearly limitations of this work, some of which we are currently exploring. Note that L-nets are added greedily. That is, if adding an L-net helps, then we keep it and try another. If the number of hidden variables is large and their interactions are complex and diffuse over the observable variables, it may not be possible for a single or even a small number of L-nets to extract any useful information. In cases like this, we would declare, wrongly, that there are simply no hidden variables. Further, we assume that k, the number of past observations provided to the network, is sufficient for recovering hidden state information. It may be that k is too small, or that the underlying system is just not amenable to this type of use of history. In both cases, we will have difficulty extracting useful information. Finally, it could be the case that an L-net learns a function of multiple hidden variables, leading us to underestimate their number. However, we would claim that in these cases the hidden variable we really care about for the purposes of predictive accuracy is the one that is a combination of more primitive hidden variables. Nonetheless, there are clearly a number of ways in which our approach can underestimate the true number of hidden variables in a system.

This work can be extended by finding other forms of the regularization penalties that will prevent the L-net from "cheating." Furthermore, it would be extremely useful to estimate the regularization weights from data rather than doing so manually. An obvious solution is to keep a portion of the data aside and algorithmically explore the space of weights (perhaps along a log scale) until weights are found for which prediction accuracy is optimized. The main goal of this regularization process is to better optimize our estimates of the hidden values. We will continue to investigate the scalability of the method to several hidden variables in different domains. Our current experiments have shown this method to be successful in domains with up to three hidden variables. There may be issues with computational speed as the number of hidden variables increases.

References

[1] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[2] J. Wnek and R. S. Michalski, "Hypothesis-driven constructive induction in aq17- hci: a method and experiments," *Machine Learning*, vol. 14, no. 2, pp. 139–168, 1994.

[3] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, 2001.

[4] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," *Advances in Neural Information Processing Systems*, vol. 2, pp. 524–532, 1990.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incom- plete data via the em algorithm," *Journal of the Royal Statistical Society Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[6] N. Friedman, "Learning belief networks in the presence of missing values and hidden variables," in *Proceedings of the 14th International Conference on Machine Learning*, pp. 125–133, Morgan Kaufmann, 1997.

[7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[8] M. L. Littman, R. S. Sutton, and S. Singh, "Predictive representations of state," in *Advances in Neural Information Processing Systems*, vol. 14, pp. 1555–1561, MIT Press, 2002.

[9] R. L. Rivest and R. E. Schapire, "Diversity-based inference of finite automata," *Journal of the ACM*, vol. 41, no. 3, pp. 555–589, 1994.

[10] M. P. Holmes and C. L. Isbell, "Looping suffix tree-based inference of partially observable hidden state," in *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, pp. 409–416, ACM, New York, NY, USA, June 2006.

[11] A. Mccallum, "Instance-based state identification for reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 7, pp. 377–384, MIT Press, 1994.

[12] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[13] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models," in *NIPS*, pp. 1441–1448, MIT Press, 2006.
[14] K. A. Bollen, "Latent variables in psychology and the social sciences," *Annual Review of Psychology*, vol. 53, pp. 605–634, 2002.
[15] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1998.
[16] F. Takens, "Detecting strange attractors in turbulence," *Lecture Notes in Mathematics*, pp. 366–381, 1981.
[17] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '90)*, vol. 3, pp. 21–26, June 1990.
[18] N. Lorenz, "Deterministic non-periodic flows," *Journal of Atmospheric Science*, 1963.
[19] A. Frank and A. Asuncion, "UCI machine learning repository," 2010.
[20] S. Ray and T. Oates, "Discovering and characterizing hidden variables in streaming multivariate time series," in *Proceedings of the 9th International Conference on Machine Learning and Applications (ICMLA '10)*, pp. 913–916, IEEE Computer Society, 2010.
[21] J. Bergstra and Y. Bengio, "Slow, decorrelated features for pretraining complex cell-like networks," in *NIPS*, 2009.
[22] L. R. Rabiner and B. H. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.