Proceedings of International Joint Conference on Neural Networks, San Jose, California, USA, July 31 – August 5, 2011

Temporal Nonlinear Dimensionality Reduction

Mike Gashler and Tony Martinez

Abstract— **Existing Nonlinear dimensionality reduction (NLDR) algorithms make the assumption that distances between observations are uniformly scaled. Unfortunately, with many interesting systems, this assumption does not hold. We present a new technique called *Temporal NLDR* (TNLDR), which is specifically designed for analyzing the high-dimensional observations obtained from random-walks with dynamical systems that have external controls. It uses the additional information implicit in ordered sequences of observations to compensate for non-uniform scaling in observation space. We demonstrate that TNLDR computes more accurate estimates of intrinsic state than regular NLDR, and we show that accurate estimates of state can be used to train accurate models of dynamical systems.**

I. Introduction

Nonlinear dimensionality reduction (NLDR) algorithms operate on an unordered set of high-dimensional observation vectors, $\mathbf{Y} = \langle \mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n \rangle$. They compute a corresponding set of low-dimensional vectors, $\mathbf{X} = \langle \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \rangle$, such that each $\mathbf{x}_i$ is a low-dimensional representation of $\mathbf{y}_i$. $\mathbf{x}_i$ may be thought of as a representation of the intrinsic values, or state, from which the corresponding observation $\mathbf{y}_i$ was derived.

In order to compute $\mathbf{X}$, most NLDR algorithms make the assumption that the pair-wise distances between neighboring points in $\mathbf{Y}$ will be approximately proportional to the corresponding pair-wise distances in $\mathbf{X}$. We refer to this as *the assumption of proportional distances*. Unfortunately, in many interesting cases, this assumption does not hold. For example, consider a robot which uses a camera to navigate within a building. Each $\mathbf{y}_t \in \mathbf{Y}$ is a high-dimensional vector of pixel values obtained from the robot's camera. When the robot moves, the amount of change in the observation is not only affected by the amount of movement, but also by the distance between the camera and the objects that are in view. Further, the color and texture patterns of the objects in view may cause observational changes to be scaled higher or lower, independent of the actual change in state. Partial occlusions may further exacerbate this problem. Because the assumption of proportional distances does not hold in many real-world systems, traditional NLDR algorithms are not suitable for estimating the intrinsic state of these systems.

We present a new technique called *Temporal NLDR* (TNLDR), which utilizes the additional information found in sequences of observations to perform nonlinear dimensionality reduction without making the assumption of proportional distances. We show that TNLDR is effective at estimating the intrinsic state of systems from high-dimensional observations, where traditional NLDR algorithms fail. We also demonstrate that an accurate estimate of intrinsic system state is useful for training a model of the system, a process also know as nonlinear black-box system identification.

Mike Gashler and Tony Martinez are with the Department of Computer Science, Brigham Young University, Provo, Utah, U.S.A. (email: {mike, martinez}@axon.cs.byu.edu).

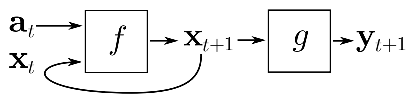

Fig. 1. A model of a dynamical system. f is the transition function of the system. g is the observation function. t specifies a time-step. $\mathbf{a}$ is an action vector, which may be discrete or continuous. $\mathbf{x}$ is a vector of continuous values that represent state. $\mathbf{y}$ is a high-dimensional observation vector.

TNLDR operates using more information than regular NLDR. TNLDR assumes that $\mathbf{Y}$ is an ordered sequence of observations occurring at regular time intervals. Additionally, a sequence of actions, $\mathbf{A} = \langle \mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n \rangle$, is given to specify the control inputs given to influence the dynamical system. By using this additional information, TNLDR is able to compute pair-wise distances between observations in a manner that is independant of the local scaling of observations, and then use existing methods to estimate system state.

TNLDR is designed to reduce the dimensionality of the output of dynamical systems. A general model of a dynamical system is given in Figure 1. At any given time, t, the output, $\mathbf{y}_t$, can be observed to provide information about the system. We assume that $\mathbf{y}_t$ is a high-dimensional vector of continuous values. $\mathbf{y}_t$ is not directly a function of the current input $\mathbf{a}_t$. Rather, $\mathbf{y}_t$ is a function of the hidden state, $\mathbf{x}_t$, which may be considered to be an aggregation of the initial state, $\mathbf{x}_0$, and all of the inputs, or actions, that were previously applied to the system, $\langle \mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{t-1} \rangle$. Actions may be continuous or discrete. At each time-step, the system transitions to a new state, which is determined by the current state and the action which is applied to the system. Thus, traditional supervised learning algorithms, which assume the output is a direct function of the input, are not sufficient for modeling dynamical systems. Dynamical systems can be challenging to model because the sequence of states, $\mathbf{X} = \langle \mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{n-1} \rangle$, is not directly observable. TNLDR, however, can greatly simplify the task of modeling a dynamical system by estimating a sequence of states, $\mathbf{X}$, that corresponds with the sequence of actions, $\mathbf{A}$, applied to the system, and the sequence of observations, $\mathbf{Y}$, that come out from the system. Thus, TNLDR estimates $\mathbf{X}$, from $\{\mathbf{A}, \mathbf{Y}\}$. In the robot example, if $\mathbf{A}$ is a sequence of n actions from the set {turn left, turn right, move forward, back up}, and $\mathbf{Y}$ is the video of n frames from the robot's camera, then

TNLDR might use this information to estimate a sequence of vectors that represent the position and orientation of the robot at each of the n time-steps.

In order for TNLDR to compensate for non-uniform local scaling in observation space, it needs a mechanism to estimate how distances are scaled at each position. Therefore, a regression model, h, is trained with each $\langle \mathbf{y}_t, \mathbf{a}_t \rangle \rightarrow (\mathbf{y}_{t+1} - \mathbf{y}_t)$, where $t < n$. h gives an estimate of how the observation vector will change when a particular action is applied at any position in observation space. The magnitude of this predicted change corresponds with the local scaling of observations. Typically, NLDR algorithms compute the pair-wise distance between two observations, $\mathbf{y}_i$ and $\mathbf{y}_j$, as $||\mathbf{y}_j - \mathbf{y}_i||$. This approach is simple, but it does not compensate for local scaling. TNLDR computes this distance by finding the most direct path of actions from $\mathbf{y}_i$ to $\mathbf{y}_j$, using h to estimate the effect of each candidate action at every step along the path. The distance between $\mathbf{y}_i$ and $\mathbf{y}_j$ is the number of actions in the path, or the number of time-steps that separate the two observations. This time-based distance metric is better-suited for analyzing the high-dimensional observations from dynamical systems because it is independent of the factors that cause observations to be scaled disproportionately with state. TNLDR uses this time-based distance metric to choose local neighborhoods, and to compute the pair-wise distances, $\mathbf{D}$, between them, where each $d_{ij} \in \mathbf{D}$ is the time-based distance between $\mathbf{y}_i$ and $\mathbf{y}_j$. TNLDR then uses a regular NLDR to compute $\mathbf{X}$ from $\mathbf{D}$. TNLDR may be summarized at a high-level as:

1. Train a regression model, h, with each $\langle \mathbf{y}_t, \mathbf{a}_t \rangle \rightarrow (\mathbf{y}_{t+1} - \mathbf{y}_t)$, where $t < n - 1$.
2. Use h to identify neighboring observations, and to compute the pair-wise distances, $\mathbf{D}$, between neighboring observations.
3. Use a regular NLDR algorithm to compute $\mathbf{X}$ from $\mathbf{D}$.

TNLDR is designed to analyze high-dimensional observations from dynamical systems. This may be useful, for example, because it will enable a system to be monitored using a general-purpose optical camera, which produces high-dimensional information, instead of problem-specific sensory devices.

We report results from several experiments that demonstrate the effectiveness of TNLDR. We compare TNLDR with existing NLDR algorithms, and show that TNLDR obtains better results. We also show that TNLDR can be used to facilitate the training of models of dynamical systems that have external controls. We compare models trained by TNLDR with those trained using conventional methods and show that the models trained with TNLDR are more accurate. We show that TNLDR enables other regression models, besides neural networks, to be used in a recurrent manner for modeling dynamical systems. We also demonstrate that a model trained with TNLDR is sufficiently accurate to facilitate planning in isolation from the system.

This paper is laid out as follows. In Section II we give an overview of related work. In Section III we describe TNLDR in detail. Section IV reports analysis and validation of TNLDR and the related SEIT method. Finally, Section V summarizes the contributions of this paper.

II. Related Work

Many works have been presented in the last decade related to NLDR [1], [2], [3], [4], [5], [6], [7], [8]. In our experiments, we use Isomap [1] and Breadth-first Unfolding with TNLDR, although other NLDR algorithms are suitable as well. The primary focus of TNLDR is for analyzing high-dimensional nonlinear observations from dynamical systems, especially when an estimate of state is required. The notion of using dimensionality reduction to estimate state has been used previously, particularly for the application of robot tracking [9], [10], [11], [12], [13]. To our knowledge, however, no dimensionality reduction technique has yet tried to specifically make use of the additional information that is found in sequences of observations.

In order to demonstrate the effectiveness of TNLDR, we use TNLDR to train models of dynamical systems. Most work to date related to modeling nonlinear dynamical systems involves recurrent neural networks. Existing methods for training the weights of recurrent neural networks can be broadly divided into two categories: Those based on nonlinear global optimization techniques, and those based on descending a local error gradient. Perhaps the most common nonlinear global optimization technique for training recurrent neural networks is evolutionary optimization [14], [15], [16]. Unfortunately, in practice, evolutionary optimization tends to be extremely slow, and it is unable to yield good results with many difficult problems [17], [15]. Evolutionary optimizers are particularly susceptible to problems where an error surface exhibits narrow channels. The optimizer will typically become stalled as it waits to find a lucky vector that falls within the narrow region of improved vectors. Gradient-based methods that converge to a local optimum offer much faster convergence than optimization techniques that seek the global optimum. Perhaps the most popular of the gradient-based techniques for recurrent networks is Backpropagation Through Time (BPTT) [18]. Another common gradient-based method is Real-Time Recurrent Learning (RTRL) [19]. Although gradient-based methods converge faster than global optimization methods, they are susceptible to problems with local optima. With feed-forward neural networks, local optima is often considered to be an insignificant problem since many local optima occur near relatively good regions of the weight space. With recurrent neural networks, however, local optima is a much more significant problem [20]. The feedback which comes through the recurrent connections can create fluctuating and chaotic responses in the error surface, causing local optima to occur both frequently and in poor locations of the weight space. By contrast, the NLDR component of TNLDR is designed specifically for estimating intrinsic state without being subject to problems with local optima. We show that this naturally leads to better models than can be obtained using BPTT and other methods.

TNLDR may be loosely comparable with the extended Kalman filter (EKF), since both are used to estimate the hidden state of a system. The EKF, however, utilizes a non-linear model of system dynamics, provided by the user, to estimate state. By contrast, TNLDR estimates state without requiring a model of the system to be supplied. Thus, the state estimated by TNLDR may be used to train a non-linear model of system dynamics.

III. The TNLDR algorithm

TNLDR is described in pseudo-code in Figure 2. We now describe each step in the algorithm.

Step 1: The first step of TNLDR is to train a supervised learning regression algorithm, h, with each $\langle \mathbf{y}_t, \mathbf{a}_t \rangle \rightarrow (\mathbf{y}_{t+1} - \mathbf{y}_t)$, where $t < n - 1$. In our implementation, we use a 1-nearest-neighbor model for h, but other algorithms may be suitable as well.

Step 2: A path of actions, $\mathbf{p}(i, j)$, is estimated from each $\mathbf{y}_i$ to $\mathbf{y}_j$, where $j \neq i$, $j < n$, and $i < n$. In our implementation, we use the simple greedy approach described with pseudo-code in Figure 3 to find this path, but other path search algorithms may be suitable as well. Our simple greedy path search constructs a set, $\mathcal{B}$, of all the unique actions in $\mathbf{A}$. At each step, it removes all actions from $\mathcal{B}$ that are not positively correlated with the remaining difference in observations. This ensures that it does not find paths that spiral inward or overshoot and then backtrack. It always chooses the action that is most positively correlated with the remaining difference, which is not necessarily the action that advances closest to the goal. This approach is more robust when it is not known how the axes are locally scaled with respect to each other in observation space. It will take at most one step before recomputing the predicted change in observations. Fractional steps are permitted when it is less than one time-step away from the goal.

The path returned by this greedy algorithm is a vector of the number of times that each action is performed on the estimated most-direct path from $\mathbf{y}_i$ to $\mathbf{y}_j$. This is a lossy representation of the path in that it does not represent the order in which the actions are performed. That information is not needed hereafter, so this is a sufficient representation.

Because this is a greedy technique, it is possible that the search for a path from $\mathbf{y}_i$ to $\mathbf{y}_j$ may become stuck in a local optimum. This condition is detected if more than a ratio of λ of the gap between $\mathbf{y}_i$ to $\mathbf{y}_j$ is unexplained by the estimated path. When this occurs, the path is rejected as an invalid estimate, and $\mathbf{y}_j$ is determined to not be a neighbor of $\mathbf{y}_i$. If the observation manifold exhibits local linearity, which is typically assumed, then short paths will not encounter local optima. Since long paths will be rejected anyway, there is little (if any) value in fine-tuning the value of λ. The only significant case when this check has any effect is when a very distant point begins close to a local optimum. In such cases, the estimated path length will be close to zero, and nearly all of the distance between $\mathbf{y}_i$ to $\mathbf{y}_j$ will remain unexplained by the path. Thus, a wide range of values for λ yields nearly identical results. We use $\lambda = 0.2$ in all of our experiments.

Typically NLDR uses one of two common techniques for determining local neighborhoods. The most common technique is to compute the k-nearest Euclidean distance neighbors of every $\mathbf{y}_t \in \mathbf{Y}$. A less-popular technique is to choose all points that fall within a distance of ϵ to be neighbors. This approach is less popular because it requires a problem-specific value for ϵ. A good value can be difficult to determine since distances in observation space depend on the nature of the observations. Since TNLDR uses a problem-independent time-based distance metric, however, it is possible to intuitively select a value for ϵ that will be suitable with many different problems. Each $\mathbf{y}_j$ is determined to be a neighbor of $\mathbf{y}_i$ if $||\mathbf{p}(i, j)|| \leq \epsilon$. In our implementation, we use the value $\epsilon = 2$. Intuitively, this means that some observation is determined to be a neighbor of the current observation if it can be reached by performing two or fewer actions. This creates a local neighborhood size that is suitable for most problems, and is robust even when some regions of the context space are sampled more heavily than others, which is typical with random walks.

If the actions are continuous, then $||\mathbf{p}(i, j)||$ is computed as the length of the path, or Manhattan magnitude. If the actions are discrete, then $||\mathbf{p}(i, j)||$ is the Euclidean magnitude of $\mathbf{p}(i, j)$. It is computed this way because the NLDR algorithm used in the next step will assume a continuous space while computing the context embedding, and will seek to preserve the Euclidean distance between points.

Step 3: $\mathbf{X}$ is computed by using an NLDR algorithm with the table of normalized neighbor distances, $\mathbf{D}$. Some NLDR algorithms that inherently support custom distance metrics include: Isomap [1], Local Multidimensional Scaling [7], and Breadth-first Unfolding . Some other existing NLDR algorithms compute distances internally, usually using Euclidean distance, and do not explicitly support custom distance metrics. These NLDR algorithms may need to be modified somewhat to be suitable for use with TNLDR.

IV. Experimental Validation

This section reports results from experiments designed to validate the utility of TNLDR. Section IV-A compares the state estimates of TNLDR with those of the corresponding NLDR algorithms. Section IV-B demonstrates the use of TNLDR to build a more accurate model of a dynamical systems than existing methods. Section IV-C demonstrates that TNLDR can be used to build models of dynamical systems using recurrent versions of arbitrary regression models. Section IV-D demonstrates that a model trained by TNLDR is sufficiently accurate to facilitate planning.

A. State Estimation

We used a simulated system involving a virtual crane with a boom and a ball that hangs from a cable. There were 4 actions associated with this system: {rotate right (yaw-wise), rotate left, extend the length of the cable, shorten the cable}. A ray-tracer was used to generate 64×48 pixel 3-channel observation images of this system, such that each observation was a 9216-dimensional vector. We generated a sequence of

```
function TNLDR($\mathbf{Y}, \mathbf{A}$)
        Let $n = |\mathbf{Y}| = |\mathbf{A}| =$ the length of time represented in the training data.
        Let $\mathbf{D}$ be a neighbor-distance table, where each $d_{ij} \in \mathbf{D}$ represents the
            distance between neighboring observations $\mathbf{y}_i$ and $\mathbf{y}_j$.
        $\epsilon \leftarrow 2$
    1.  Train a regression model, $h$, with each $\langle \mathbf{y}_t, \mathbf{a}_t \rangle \rightarrow (\mathbf{y}_{t+1} - \mathbf{y}_t)$, where $t < n - 1$.
    2.  for each $\mathbf{y}_i \in \mathbf{Y}$:
            for each $\mathbf{y}_j \in \mathbf{Y}$, $j \neq i$:
                $\mathbf{p}(i,j) \leftarrow$FindPath($\mathbf{y}_i, \mathbf{y}_j$), where FindPath is defined in Figure 3
                if $||\mathbf{p}(i,j)|| < \epsilon$ then $d_{ij} = ||\mathbf{p}(i,j)||$ else $\mathbf{y}_j$ is not a neighbor of $\mathbf{y}_i$
    3.  Use an NLDR algorithm to compute a sequence of context vectors, $\mathbf{X}$, from $\mathbf{D}$.
```

Fig. 2. Pseudo-code for TNLDR.

```
function FindPath($\mathbf{y}_i, \mathbf{y}_j$)                                    Comments
$r \leftarrow ||\mathbf{y}_j - \mathbf{y}_i||$                                       Measure the initial residual
$\mathcal{B} \leftarrow$ the set of unique actions in $\mathbf{A}$                    Make a set of candidate actions
$\mathbf{p}(i,j) \leftarrow \mathbf{0}^{|\mathcal{B}|}$                              Start with empty path
while $|\mathcal{B}| > 0$ :                                                          While there are useful actions
    for each $\mathbf{b} \in \mathcal{B}$:                                           Prune counter-productive actions
        if $h(\mathbf{y}_i, \mathbf{b}) \cdot (\mathbf{y}_j - \mathbf{y}_i) \leq 0$  If $\mathbf{b}$ is not positively correlated
            remove $\mathbf{b}$ from $\mathcal{B}$                                   Do not try action $\mathbf{b}$ again
    if $|\mathcal{B}| > 0$ :                                                         If there are still useful actions
        $\mathbf{a} \leftarrow \text{argmax}_{\mathbf{b} \in \mathcal{B}} \frac{h(\mathbf{y}_i, \mathbf{b})}{||h(\mathbf{y}_i, \mathbf{b})||} \cdot (\mathbf{y}_j - \mathbf{y}_i)$    Find the best-correlated action
        $s \leftarrow \min(1, \frac{h(\mathbf{y}_i, \mathbf{a}) \cdot (\mathbf{y}_j - \mathbf{y}_i)}{||\mathbf{y}_j - \mathbf{y}_i||^2})$    Compute fractional action
        $\mathbf{y}_i \leftarrow \mathbf{y}_i + s * h(\mathbf{y}_i, \mathbf{a})$    Step closer to $\mathbf{y}_j$
        $p_{\mathbf{a}}(i,j) \leftarrow p_{\mathbf{a}}(i,j) + s$                     Update the count for action $\mathbf{a}$
if $||\mathbf{y}_j - \mathbf{y}_i|| < \lambda * r$ then return $\mathbf{p}(i,j)$ else return null    Reject very poor estimates
```

Fig. 3. Pseudo-code for a greedy algorithm that estimates $\mathbf{p}(i,j)$, where $p_{\mathbf{a}}(i,j)$ is the number of times that action a is performed on the estimated most-direct path from $\mathbf{y}_i$ to $\mathbf{y}_j$. (Fractional counts are permitted.) The value $\lambda = 0.2$ is suitable in almost all cases.

4000 random actions, $\mathbf{A}$, and applied them to this system to obtain the corresponding 4000 observation images, $\mathbf{Y}$

Figure 4A shows the result of using principal component analysis, a linear dimensionality reduction technique, to reduce $\mathbf{Y}$ into two dimensions. This visualization shows that the actions have a non-uniform impact on the observations. In most cases, changing the yaw angle of the crane has a large impact on the system in observation space, while changing the length of the cable has a relatively small impact in observation space. This is manifest in the formation of small string-like clusters in the PCA plot. Each "string" is composed of observations with the same yaw-angle, but different cable lengths. Effective neighbor-finding is difficult in this space for two reasons: First, Euclidean-distance will tend to pick only neighbors with the same yaw-angle, since the cable-length has a lesser impact on observations. Second, the random walk samples the space non-uniformly. In order to facilitate NLDR, local neighborhoods must be transitively connected across the entire manifold, but in order to achieve this using Euclidean-distance, the neighborhoods must be so large that undesirable topological structures will be represented in the neighborhoods.

Figure 4B shows these observations reduced using Isomap with neighborhoods of size 48. The large number of neighbors was necessary to produce transitive connectivity. Figure 4C shows results with Breadth-first Unfolding (BFU) with neighborhoods of size 48. For comparison, Figure 4D shows the actual hidden state of the system. Figure 4E shows results with TNLDR using Isomap. Figure 4F shows results with TNLDR using BFU. TNLDR improved the results from both algorithms. When better distances go into an NLDR algorithm, better state estimates comes out.

B. Modeling Dynamical Systems

Perhaps the best way to demonstrate the utility of TNLDR is to demonstrate its use in training a model of a dynamical system. This is done with a simple method that we call *State Estimate Induced Training* (SEIT). The steps of SEIT are:

1. Use TNLDR to compute $\mathbf{X}$ from $\{\mathbf{Y}, \mathbf{A}\}$.
2. Train a regression model, f, with each $\langle \mathbf{a}_t, \mathbf{x}_t \rangle \rightarrow \mathbf{x}_{t+1}$, where $t < n - 1$.
3. Train a regression model, g, with each $\mathbf{x}_t \rightarrow \mathbf{y}_t$, where $t < n$.
4. Return $\langle \mathbf{x}_0, f, g \rangle$. (Note that $\mathbf{x}_0 \in \mathbf{X}$.)

SEIT uses TNLDR to divide the recurrent model shown in Figure 1 into two simpler parts, f and g, which may each be trained as a static model. SEIT computes a model of a dynamical system, $\langle \mathbf{x}_0, f, g \rangle$, where $\mathbf{x}_0$ is the initial estimate of

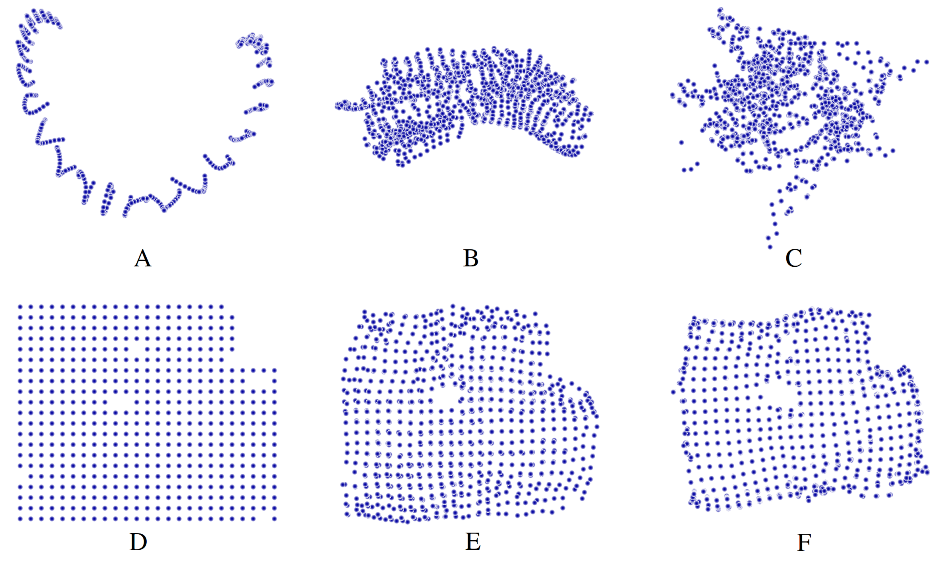

Fig. 4. A) A PCA projection of $\mathbf{Y}$ into 2 dimensions. B) An Isomap projection of $\mathbf{Y}$. C) A Breadth-first Unfolding projection of $\mathbf{Y}$. D) The actual hidden states visited by the random walk. E) A TNLDR projection of $\mathbf{Y}$ using Isomap. F) A TNLDR projection of $\mathbf{Y}$ using Breadth-first Unfolding. TNLDR estimates the state of dynamical systems better than regular NLDR algorithms.

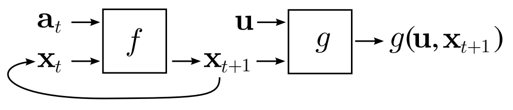

Fig. 5. High-dimensional observations may be parameterized with a vector, $\mathbf{u}$, such that the model predicts only the portion of $\mathbf{y}_{t+1}$ specified by $\mathbf{u}$. For example, if $\mathbf{y}_{t+1}$ is an image, then $g(\mathbf{u}, \mathbf{x}_{t+1})$ could be a prediction of the three channel values (red, green, and blue) of pixel $\mathbf{u}$ in that image.

state, f is a transition function which specifies how the state changes at each time-step, and g is an observation function which specifies the relationship between observations and state. If f and g are both modeled with feed-forward neural networks, then the model as a whole is an Elman recurrent neural network. SEIT provides a convenient mechanism for validating the effectiveness of TNLDR, because it may be compared empirically with existing methods for training Elman recurrent neural networks, such as Backpropagation Through Time (BPTT), evolutionary optimization, simulated annealing, etc.

In contrast with BPTT, SEIT has several advantages. BPTT must simultaneously train f and g so that these two functions will learn to cooperate. This makes it highly susceptible to problems with local optima. Also, BPTT it is not suitable for training some models, such as support vector machines or regression trees. By contrast, SEIT determines how f and g should cooperate before their training begins, and encodes this information in $\mathbf{X}$. Thus, f and g can be trained independently, and they are trained against a stationary target. The NLDR component of TNLDR naturally avoids local optima in the representation of $\mathbf{X}$. Further, SEIT can be used to train arbitrary regression models to operate in a recurrent manner.

We trained a model of the crane system, $\langle \mathbf{x}_0, f, g \rangle$, using SEIT. We modeled f using a feed-forward neural network with 6 inputs (4 to represent the actions, and 2 from recurrent connections), one hidden layer of 4 sigmoid units, and 2 output units. We modeled g using a feed-forward neural network with 4 inputs (2 from f, and 2 to parameterize the image pixel as shown in Figure 5), and two hidden layers. The first hidden layer (in feed-forward order) contained 15 units, and the second hidden layer contained 30 units. g had 3 output units (for the three color channels). Together, f and g form a recurrent neural network with 668 weights. We tested five algorithms for training this network from $\{\mathbf{Y}, \mathbf{A}\}$. At 120-second intervals during training, we measured the root-mean-squared predictive accuracy of each model, averaged over 5 validation sequences, each of consisting of 40 random actions and the corresponding observations. Figure 6 shows a comparison of the results obtained by each algorithm. SEIT required almost 500 seconds to compute $\mathbf{X}$ using TNLDR, and to train f. Results for SEIT are only shown during the training of g. (With SEIT, f and g could be trained in parallel, but we did not utilize this advantage in this experiment.) Three of the algorithms all arrived at the poor solution of always predicting a completely white image. With this problem, there is a broad locally-convex region around this solution because the significant majority of the pixels in the true observation associated with every state is white. It may be that the evolutionary optimizer would eventually find its way out of this local optimum, but even if it does, this is a very inefficient solution. We ran that algorithm for 7 additional hours, but it did not manage to break out in that time. Backpropagation Through Time was the closest competitor with SEIT. Unfortunately, it appears to have quickly found a local optimum from which it never managed to escape. SEIT produced the best results by a

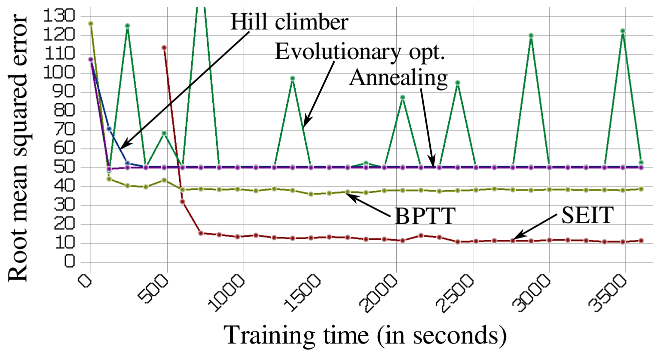

Fig. 6. Predictive error averaged over 5 unique validation sequences, each with 40 actions and observations, was measured at 120 second intervals during training with 5 algorithms. Three algorithms converged to always predict a blank image. BPTT did better. SEIT gave the best results. Approximately the first 500 seconds were required to compute $\mathbf{X}$ and to train the transition function, so results are shown for SEIT during training of the observation function.

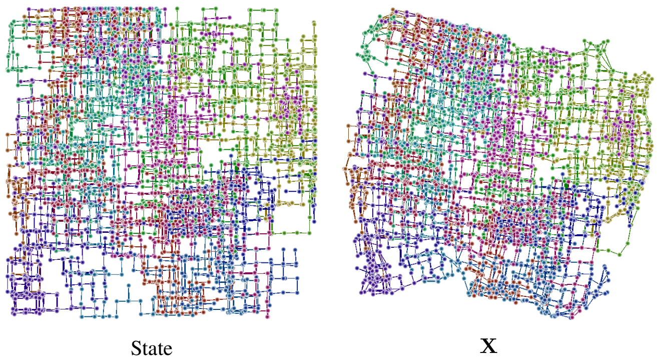

Fig. 7. A comparison of the true hidden state from the noisy crane system, and the estimate of state, $\mathbf{X}$, computed by the first three steps of SEIT. To assist a visual comparison of the structure, each point is shown with a spectrum color according to its position in the sequence, and lines are also shown to indicate transitions.

significant margin.

Next, we modified the crane system to add random noise to each hidden state variable at every transition. The noise was drawn from a Normal distribution with a deviation equal to 5% of the magnitude of the change in state. Random noise was also added to all three channels of every pixel in the observation. This noise was also drawn from a Normal distribution with a deviation equal to 5% of the supported range in channel values. Figure 7(left) shows a plot of the actual hidden state of the system, and Figure 7(right) shows $\mathbf{X}$ as estimated by TNLDR using Breadth-first Unfolding. Despite the noise in observations, TNLDR was still able to estimate a good representation of the system state. We note that for training a model of the system, $\mathbf{X}$ does not need to be strictly equivalent to the hidden state, as long as f and g are able to compensate for differences.

Figure 8 shows a comparison of the actual observations from the noisy system, with predictions from SEIT and BPTT, over a test sequence of 200 random actions. The observation sequence was predicted from only the test actions, without any feedback from the system. SEIT predicted each frame clearly, while BPTT made blurry and ambiguous predictions. With BPTT, the interplay between f and g during training caused the internal state estimate to fall into a local optima. By contrast, SEIT did a better job of directing how f and g should mutually behave by computing $\mathbf{X}$.

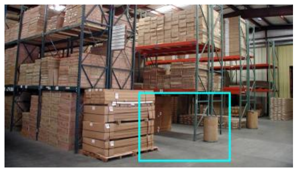

Fig. 10. A robot's observations were simulated using a sliding and scaling window over an image of a warehouse.

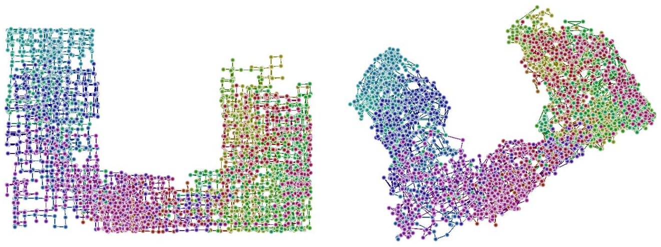

Fig. 11. Left: A plot of the hidden states through which the warehouse system passed while generating the training observations. Right: A plot of $\mathbf{X}$ as computed by the first step of SEIT. Color is used to indicate the position of points in the ordered sequence.

C. Decision Tree Model

In order to demonstrate that TNLDR enables the training of a recurrent version of arbitrary regression models, not just recurrent neural networks, we trained a decision tree to model both f and g. Figure 9 shows a comparison of actual and predicted observations with this model. Existing methods for training recurrent neural networks, such as BPTT, are not able to train models based on decision trees.

D. Path Planning

We created another system using the image of a warehouse shown in Figure 10. The observations for this system were taken from a small window within this larger image. The system was equipped with 4 actions, where two of the actions slide the window left or right, and the other two actions zoom in or out by changing the size of the window. The window is capable of having a continuous position and size, so we used linear interpolation to generate a 64×48 pixel observation image that spans the windowed region of the larger image. The observations of this system were designed to be similar to those of a robot that navigates within a warehouse. As with the noisy crane system, we added Gaussian noise to both the transitions and observations, with the same deviations used in that system.

Additionally, we blocked the system from being able to enter a square region of its state space. A robot, for example, may be blocked by a large object from entering

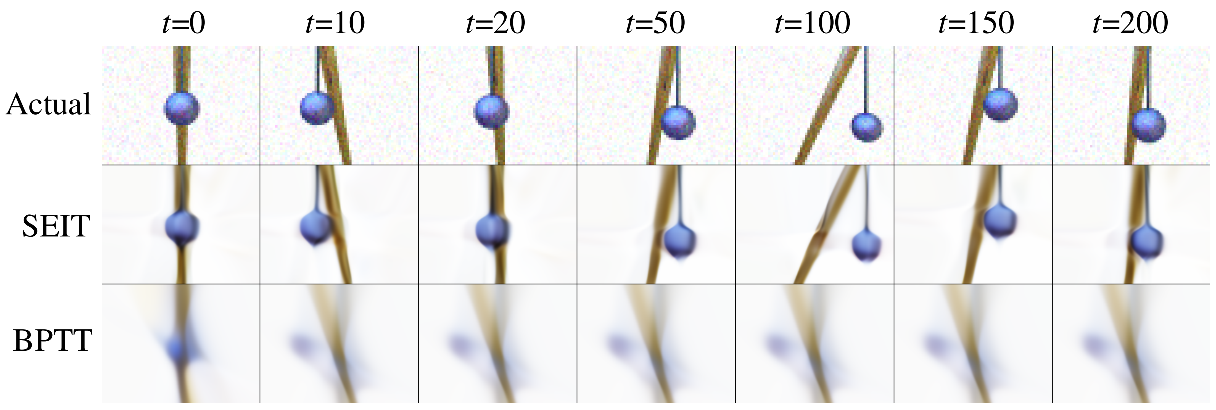

Fig. 8. Samples of actual and predicted observations for a test sequence of actions, unrelated to $\mathbf{A}$. Predictions were made by the recurrent models from the test actions, without any feedback from the system. SEIT made accurate and clear predictions, while BPTT made blurry predictions.

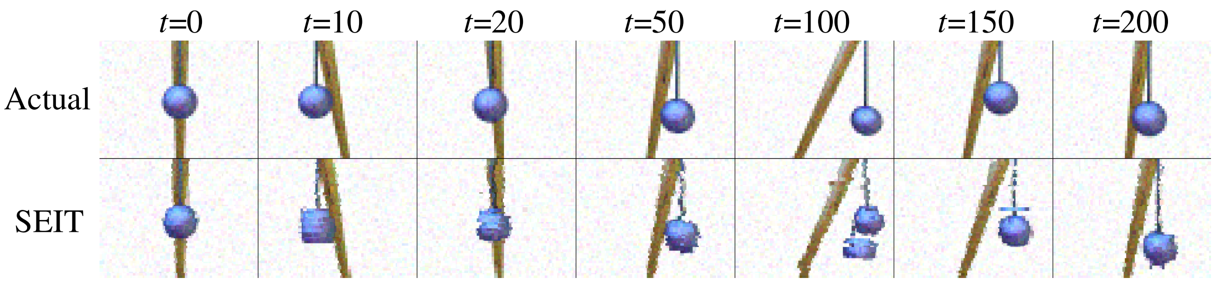

Fig. 9. Sample predictions from a model using decision trees for f and g. Existing algorithms are only suitable for training recurrent neural networks. SEIT can train arbitrary recurrent models.

certain regions of its state space. Such a robot may need to learn to model its environment even though it is unable to obtain observations from those regions of its state space. We generated a new training sequence of 4000 random actions, and obtained a corresponding sequence of observations from this warehouse system. Figure 11 shows a comparison of the actual hidden state produced by this system, and the estimate of state computed by the first step of SEIT.

We modeled this system with a recurrent neural network, where f had one hidden layer of 4 units, and two context units, and g had two hidden layers. The first hidden layer in feed-forward order had 20 units, and the second hidden layer had 100 units. We used more units in g with the warehouse system because its observations contained more detail than the crane system.

Figure 12 shows a comparison of results with this problem using several training algorithms. Accuracy was measured by averaging over 5 validation sequences of 40 actions and observations containing both observation noise and transition noise. The transition noise has a particularly significant impact on predictions because it accumulates in the state over time, while the observation noise affects only the current observation. Even under these conditions, SEIT was able to give the best results of any of the algorithms, using either Isomap or Breadth-first Unfolding.

Next, using only the trained neural network model to predict observations, a human oracle selected a sequence of

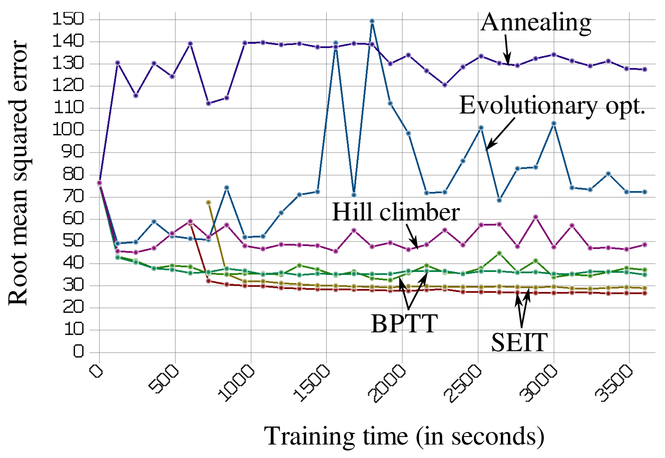

Fig. 12. Predictive error with validation data was measured at 120 second intervals during training with several algorithms. SEIT gave the best results. The choice of NLDR algorithm used with SEIT made little difference.

actions that would cause the simulated robot to visit certain locations within its environment. The intrinsic states in this planned path are shown superimposed over a plot of $\mathbf{X}$ in Figure 13(left). We then executed the planned sequence of actions with the actual system. Figure 13(right) shows the actual hidden state values through which the system passed as it followed the sequence of actions, superimposed over the actual state values that correspond with the training

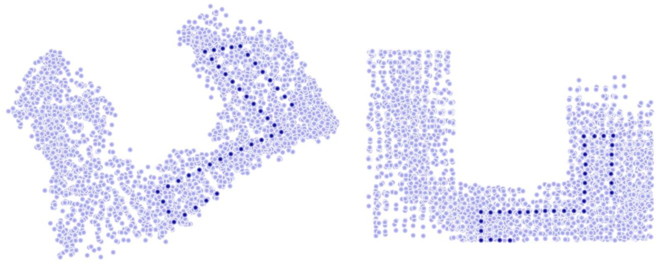

Fig. 13. Left: A path of planned context values chosen by a human based on predicted observations from the neural network model of the warehouse system. Right: The path of actual state values through which the system passed when the planned actions were applied to the system.

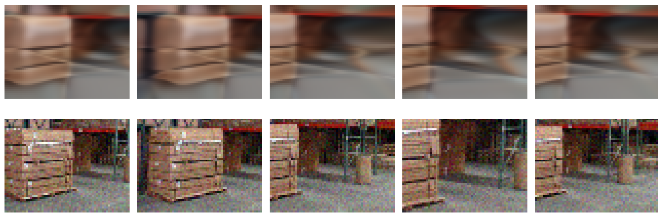

Fig. 14. Top: Predicted observations from a planned path made using only a model of the warehouse system. Bottom: The actual observations made when the planned path was executed with the warehouse system.

observations that were originally used to train the model. Figure 14(top) shows predicted observations at five points along the planned path. Figure 14(bottom) shows the actual observations at those points when the plan was executed on the system. This experiment demonstrates that the trained model represented the system with sufficient accuracy that it could facilitate planning in isolation from the system.

V. Conclusions

We presented a new technique called TNLDR, which reduces the dimensionality of observations from a dynamical system to recover an estimate of the system state. Compared with regular NLDR, TNLDR uses the additional information found in sequences of observations to remove the assumption that distances in state space are proportional to distances in observation space. Because TNLDR removes this assumption, it can compute accurate estimates of state, even when various factors cause observations to be scaled non-uniformly. TNLDR has significant potential to lead to further innovations because it extends existing NLDR techniques to make them suitable for use in estimating the state of dynamical systems from high-dimensional observations.

We used a simulated crane system to demonstrate that TNLDR recovers better estimates of state than existing NLDR techniques. We also demonstrated that TNLDR leads to a natural method for training models of dynamical systems, called SEIT. We showed that SEIT does a better job training a recurrent neural network to model the crane system than existing methods for training recurrent neural networks. We also repeated this experiment using a system involving a simulated robot in a warehouse. We showed that SEIT is suitable for training other recurrent models besides neural networks. We also demonstrated that models trained by SEIT are sufficiently accurate to facilitate planning.

References

[1] J. B. Tenenbaum, V. de Silva, and J. C. Langford, “A global geometric framework for nonlinear dimensionality reduction,” *Science*, vol. 290, pp. 2319–2323, 2000.

[2] S. T. Roweis and L. K. Saul, “Nonlinear dimensionality reduction by locally linear embedding,” *Science*, vol. 290, pp. 2323–2326, 2000.

[3] Z. Zhang and H. Zha, “Principal manifolds and nonlinear dimension reduction via local tangent space alignment,” *SIAM Journal of Scientific Computing*, vol. 26, pp. 313–338, 2002.

[4] D. Donoho and C. Grimes, “Hessian eigenmaps: locally linear embedding techniques for high dimensional data,” *Proceedings of National Academy of Sciences*, vol. 100, no. 10, pp. 5591–5596, 2003.

[5] K. Q. Weinberger, F. Sha, and L. K. Saul, “Learning a kernel matrix for nonlinear dimensionality reduction,” in *ICML ’04: Proceedings of the Twenty-First International Conference on Machine Learning*. New York, NY, USA: ACM Press, 2004. [Online]. Available: http://dx.doi.org/10.1145/1015330.1015345

[6] E. Levina and P. J. Bickel, “Maximum likelihood estimation of intrinsic dimension,” in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 777–784.

[7] J. Venna and S. Kaski, “Local multidimensional scaling,” *Neural Networks*, vol. 19, no. 6, pp. 889–899, 2006.

[8] M. Gashler, D. Ventura, and T. Martinez, “Iterative non-linear dimensionality reduction with manifold sculpting,” in *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. Cambridge, MA: MIT Press, 2008.

[9] M. J. Black, “Eigentracking: Robust matching and tracking of articulated objects using a view-based representation,” in *International Journal of Computer Vision*, 1996, pp. 329–342.

[10] J. Crowley, F. Wallner, and B. Schiele, “Position estimation using principal components of range data,” *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 4, pp. 3121–3128 vol.4, May 1998.

[11] F. Pourraz and J. L. Crowley, “Continuity properties of the appearance manifold for mobile robot position estimation,” in *in Proceedings of the 2nd IEEE Workshop on Perception for Mobile Agents*. IEEE Press, 1998, p. 2001.

[12] S. Nayar, S. Nene, and H. Murase, “Subspace methods for robot vision,” *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 5, pp. 750–758, Oct 1996.

[13] N. Keeratipranon, F. Maire, and H. Huang, “Manifold learning for robot navigation,” *International Journal of Neural Systems*, vol. 16:5, pp. 383–392, October 2006.

[14] D. Floreano and F. Mondada, “Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot,” in *In*. MIT Press, 1994, pp. 421–430.

[15] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Deylon, P. yves Glorennec, H. Hjalmarsson, and A. Juditsky, “Nonlinear black-box modeling in system identification: a unified overview,” *Automatica*, vol. 31, pp. 1691–1724, 1995.

[16] A. Blanco, M. Delgado, and M. C. Pegalajar, “A real-coded genetic algorithm for training recurrent neural networks,” *Neural Networks*, vol. 14, no. 1, pp. 93–105, 2001.

[17] E. D. Sontag, “Neural networks for control,” in *in Essays on Control: Perspectives in the Theory and its Applications (H.L. Trentelman and*. Birkhauser, 1993, pp. 339–380.

[18] M. C. Mozer, “A focused backpropagation algorithm for temporal pattern recognition,” in *Backpropagation: Theory, architectures, and applications*, Y. Chauvin and D. Rumelhart, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1995, pp. 137–169.

[19] A. J. Robinson and F. Fallside, “The utility driven dynamic error propagation network,” Cambridge University, Engineering Department, Tech. Rep. CUED/F-INFENG/TR.1, 1987.

[20] M. Cuéllar, M. Delgado, and M. Pegalajar, “An application of non-linear programming to train recurrent neural networks in time series prediction,” *Enterprise Information Systems VII*, pp. 95–102, 2006.

[21] O. Jenkins and M. Matarić, “A spatio-temporal extension to isomap nonlinear dimension reduction,” in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 56.