Tree-Based Methods for Fuzzy Rule Extraction

Shuqing Zeng, Nan Zhang and Juyang Weng
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824-1226, USA
{zengshuq, nanzhang, weng}@cse.msu.edu

Abstract
This paper is concerned with the application of a tree-based regression model to extract fuzzy rules from high-dimensional data. We introduce a locally weighted scheme to the identification of Takagi-Sugeno type rules. It is proposed to apply the sequential least-squares method to estimate the linear model. A hierarchical clustering takes place in the product space of systems inputs and outputs and each path from the root to a leaf corresponds to a fuzzy IF-THEN rule. Only a subset of the rules is considered based on the locality of the input query data. At each hierarchy, a discriminating subspace is derived from the high-dimensional input space for a good generalization capability. Both a synthetic data set as well as a real-world robot navigation problem are considered to illustrate the working and the applicability of the algorithm.

Introduction
The fuzzy rule-based model represents a simple and powerful tool to model system dynamics by means of IF-THEN rules. Human experts’ knowledge is usually used to design those rules. This acquisition process is cumbersome, and in some cases (e.g., unknown system), expert knowledge is not available. Neuro-fuzzy approaches (Lin & Lee 1996) are introduced with the purpose of extracting the fuzzy rules, and the membership-functions automatically from measured input-output pairs. However, in neural-fuzzy approaches, the dimension of the input vector is usually small in order to be manageable.

In this paper, we propose a hierarchical fuzzy model, which extracts rules from high-dimensional data pairs. The major distinctions of the model include: First, we derive automatically discriminant feature subspaces in a coarse-to-fine manner from the high-dimensional input space. The features are most discriminative in the sense that input variables irrelevant to the output are disregarded to achieve better generalization. Second, we organize the rule base in a hierarchical way. This tree architecture recursively excludes many far-away, unrelated rules from consideration; thus, the time for inference and learning is \(O(\log(N))\), where \(N\) is the size of the rule base.

The remainder of this paper is organized as follows: Section presents the fuzzy model. We then extend it to the hierarchical form in Section . The results of the simulation and real robot experiments are reported in Section . Discussions and concluding remarks are given in Section .

Fuzzy Modeling

Fuzzy Rule-Based Systems

A fuzzy rule is a syntactic structure of the form

\[
\text{IF antecedent THEN consequent (1)}
\]

where each antecedent and consequent are well-formed fuzzy predicates. Given a system with the input vector \(x \in X \subset \mathbb{R}^n\) and the output vector \(y \in Y \subset \mathbb{R}\). In the Takagi-Sugeno (TS) model (Takagi & Sugeno 1985), the base of \(r\) fuzzy rules is represented by

\[
\text{IF } x \text{ is } A_1 \text{ THEN } y_1 = \beta_1^T z + \beta_{0,1} \\
\vdots \\
\text{IF } x \text{ is } A_N \text{ THEN } y_r = \beta_N^T z + \beta_{0,N}
\]

where \(A_1, \ldots, A_N\) denote multivariate antecedents (fuzzy predicates) defined on the universe \(\mathbb{R}^d\). Here the consequents are linear function of the dependent vector \(z\) that is the projected vector on a discriminating subspace (see Section ). The membership function of the \(i\)-th antecedent \(A_i\) is defined as

\[
A_i(x) : \mathbb{R}^d \mapsto [0, 1] 
\]

For a query input \(x\) the output of the rule-base is calculated by aggregating the individual rules contributions

\[
y = \frac{\sum_{i=1}^{N} A_i(x) y_i}{\sum_{i=1}^{N} A_i(x)}
\]

where \(y_i\) and \(A_i(x)\) denote the output and the activation level of the \(i\)-th rule, respectively.

Incremental Parameter Estimation

Two phases are involved to estimate the parameters of the \(i\)-th rule. First, in structure identification phase the fuzzy predicate \(A_i\) is determined, which will be discussed in Section . Second, in the parameter identification phase, we assume the
antecedent predicate $A_i$ is fixed and apply sequential least-square algorithm to estimate the parameters: $\beta_i$ and $\beta_{0,i}$ of the rule.

For notation simplicity we neglect rule index $i$. Consider a collection of $t$ input-output data pairs $(x_k, y_k), k = 1, 2, ..., t$ where $x_k$ is the $n$ dimensional input vector and $y_k$ denotes the scalar target output of the system. Let $z_k, k = 1, ..., t$ be extracted feature vector which we will discuss in Section .

As in locally weighted learning (LWR) (Atkeson, Moore, & Schaal 1997), we compute the following diagonal activation matrix:

$$W = \text{diag}[A(x_1) \ A(x_2) \ ... \ A(x_t)]$$

with each one corresponding to the data pair. Let $Z = \begin{bmatrix} z_1^T & \cdots & z_t^T \end{bmatrix}^T$. For computational and analytical simplicity, let $Z_e = [Z \ 1]$ and $\theta = [\beta^T \ \beta_0]^T$.

We formulate the estimation as finding the parameter $\theta$ such that the rule output, $y$, is

$$y = \theta^T z + n$$

such that the following cost function is minimized

$$(y - Z_e \theta)^T W (y - Z_e \theta).$$

where $n$ denotes a white Guassian process.

The solution of this weighted least square problem is

$$\theta = (Z_e^T W Z_e)^{-1} Z_e^T W y$$

(6)

Let the weight of the $k$-th data pair $(x_k, y_k), (k = 1, ..., t)$ is the square root of the $i$-rule’s antecedent $A(x_k)$, i.e.,

$$a_k = \sqrt{A(x_k)}$$

Each row $k$ of $X_e$ and $y$ is multiplied by the corresponding weight $a_k$ creating new variables $z_k^* = a_k z_k$ and $y_k^* = a_k y_k$. This can be done using matrix notation $Z^* = Z W^{1/2}$ and $y^* = y W^{1/2}$, where $Z^* = \begin{bmatrix} z_1^T & \cdots & z_t^T \end{bmatrix}^T$ and $y^* = \begin{bmatrix} y_1^T & \cdots & y_t^T \end{bmatrix}^T$. Therefore, Eqs. (5) and (6) become respectively

$$y = X^* \theta + n$$

(7)

and

$$\theta = (Z^*^T Z^*)^{-1} Z^*^T y^*$$

(8)

We note that Eq. (8) assumes the collection of input-output pairs are available before the estimation. This is not suitable for online identification where data pairs arriving sequentially since the inverse operation is too expensive to recompute whenever a new data comes in. However, it is possible to update $\theta$ incrementally as new data pairs are acquired.

Let us write the linear consequent part of the rule, Eq. (7) as follows

$$\begin{bmatrix} y_1^* \\ \vdots \\ y_{t-1}^* \\ y_t^* \end{bmatrix} = \begin{bmatrix} Z_{t-1}^* \\ \vdots \\ Z_1^* \end{bmatrix} \theta + \begin{bmatrix} n_1 \\ \vdots \\ n_{t-1} \\ n_t \end{bmatrix}$$

(9)

Let $\hat{\theta}^{(t)}$ denote the estimated $\theta$ of Eq. (9) after presenting the data set $X_t$. Let $P_t^{-1} = Z_t^T Z_t^*$. The recursive equations for sequentially estimating $\theta$ can be written as

$$\begin{align*}
\hat{\theta}^{(t)} &= \hat{\theta}^{(t-1)} + \gamma_t P_{t-1} z_t^* (y_t^* - z_t^T \hat{\theta}^{(t-1)}) \\
P_t &= P_{t-1} - \gamma_t P_{t-1} z_t^* z_t^T P_{t-1} \\
\gamma_t &= 1 + z_t^T P_{t-1} z_t^*
\end{align*}$$

(10)

where

$$\begin{bmatrix} y_t^* \\ z_t^* \end{bmatrix} = a_t y_t$$

(11)

The reason that we do not do regression on $x$ directly is that matrix $P_t$ in Eq. (10) is a $d \times d$ matrix, where $d$ denotes the dimension of the input vector. Computing such a matrix for input $x$ (e.g., a system with 1000 input variables) is too expensive.

### Hierarchical Fuzzy Model

In this section, we propose a hierarchical fuzzy model. The fuzzy model presented in the previous section is a flat model. All rules in the base take part simultaneously in the inference process, each to an extent proportionate to the firing value associated with its antecedent (e.g., the fuzzy predicate $A_i$ of the $i$-th rule). This poses a problem for learning since the parameters of all rules need to be updated when a new pair is presented, which is computationally too expensive.

#### Clustering

This involves successively partition the product space of input-output pairs so that at each level of the hierarchy, data pairs within the same region (cluster) are more similar to each other than those in different regions (clusters). As shown in Fig. 1, each node denotes a cluster or a rule’s antecedent in fuzzy terminology. For example, region 1 denotes the support set of $A_1$ and region 1.1 denotes the support set of $A_{11}$. A path from the root to a leaf represents a rule, i.e.,

$$\text{IF } A_{k_1}(x) \land A_{k_2}(x) \land \ldots \land A_{k_n}(x) \text{ THEN } y_i = \beta_{k_1}^T x + \beta_{0,i}$$

We observe that

$$A_{k_1}(x) \supset A_{k_1 k_2}(x) \supset \ldots \supset A_{k_1 k_2 \ldots k_n}(x)$$

Therefore the antecedent $A_{k_1}(x) \land A_{k_2}(x) \land \ldots \land A_{k_1 k_2 \ldots k_n}(x) = A_{k_1 k_2 \ldots k_n}(x)$. In addition, a cluster $A_{k_1 \ldots k_n}(x)$ not fired unless its parent $A_{k_1 \ldots k_{n-1}}(x)$ is activated. The tree structure recursively excludes many inapplicable rules from consideration, thus, the time to retrieve and update the tree for each newly arrived data point $x$ is $O(\log(N))$, where $N$ is the number of rules. This extremely low time complexity is essential for online learning with a very large rule base.

Let’s begin with the available $K$ input-output pairs. The data set to be clustered is represented as a data matrix composed from $H$ and $y$.

$$U^T = [X \ y]$$

(12)

with each column $u_k$ represents an input-output pair: $u_k = [x_k^T \ y_k]^T$, $k = 1, \ldots, K$. 

becoming a singleton or the dissimilarity within the cluster
Saul 2000), we can view informative low-dimensional pro-

definition 1 The dissimilarity measure \( d \) between the two
pairs \( i,j \), \( 1 \leq i,k \leq K \) is defined as
\[
d(u_i, u_k) = w_x \frac{\|x_i - x_k\|^2}{\sigma_x^2} + w_y \frac{\|y_i - y_k\|^2}{\sigma_y^2}
\]  \( (13) \)
where \( w_x \) and \( w_y \) are two positive weights that sum to 1:
\( w_x + w_y = 1 \); \( \sigma_x \) and \( \sigma_y \) denote estimated scalar scatter of
\( x \) and \( y \) vectors, respectively.

The hierarchical clustering algorithm begins with the en-
tire data set as a single cluster \( G \), and recursively splits one
of the existing clusters into \( q \) child clusters: \( G_1, \ldots, G_q \) in a
top-down fashion. For each iteration, we apply K-means
(see Algorithm 1) to perform the splits. We begin with
choosing the \( q \) most separated samples in \( G \) as the initial
means: \( \{u_1, \ldots, u_q\} \) and \( u_l = (\bar{x}_l, \bar{y}_l) \), \( l = 1, \ldots, q \).

The recursive splitting continues until all clusters either
becoming a singleton or the dissimilarity within the cluster
of each one from another is less than a threshold.

**Local Feature Extraction**

We note that in Eq. (2), the linear model applies on the input
vector \( x \) directly, which might contain irrelevant dimen-
sions of the input data ("curse of dimensionality"). Given
the empirical observation that the true intrinsic dimension-
ality of high dimensional data is often very low(Roweis &
Saul 2000), we can view informative low-dimensional pro-
jections of the data.

Consider an internal node \( G \) of the tree with \( q \) child
clusters in \( d \)-dimensional input space whose means are

\[ \{\bar{x}_1, ..., \bar{x}_q\} \]. Those clusters lie in an linear manifold of
dimension \( \leq q - 1 \), and if \( d \) is much larger than \( q \), this will be
a considerable drop in dimension.

We consider that two special cases of the dissimilarity
measure defined in Eq. (13). First, let \( w_x = 0 \) and \( w_y = 1 \)
and this is equivalent to supervised learning. Thus each
cluster corresponds to a class (i.e., a different output value
from that of other clusters). In locating the closest cluster,
we can ignore distances orthogonal to this subspace, since
they will contribute equally to each class. We might just as
well project the \( X \) onto this cluster-spanning subspace \( H_L \).
There is a fundamental dimension reduction, namely that we
need only consider the data in a subspace of dimension at
most \( q - 1 \). Second, let \( w_x = 1 \) and \( w_y = 0 \) corresponding
to unsupervised learning, where the clusters do not contain
label related information. The clusters spread out as much
as possible in term of variance and this amounts to finding
principle component (PCA) subspaces of the data. Other
configurations of \( w_x \) and \( w_y \) find a subspace softly combin-
ing the LDA and PCA ones.

In summary, finding the sequences of optimal subspaces
for the most discriminating features (MDF) involves the fol-
lowing steps:

- compute the \( q \times d \) matrix of clusters \( M \) and the common
covariance matrix \( W \) (for within-class covariance)
- compute \( M^* = MW^{-\frac{1}{2}} \) using the singular value decom-
position (SVD) of \( W \) (sphering the within-class covari-
ance)
- apply the Gram-Schmidt procedure on the columns of \( M^* \)
yielding \( \mathbf{v}_l \), \( l = 1, ..., L \) and \( L \leq q - 1 \).
- compute the basis of the MDF subspace \( \mathbf{v}_l = W^{-\frac{1}{2}} \mathbf{v}_l \),
\( l = 1, ..., L \).

Practically, we assume that the within-class covariance \( W =
\sigma^2 I \) where \( I \) denotes an identity matrix. This, applying

\[ \text{Algorithm 1 Cluster splitting algorithms} \]

\begin{algorithm}
1: Randomly select \( u_i \) from \( G \).
2: \textbf{for} \( k = 2, \ldots, q \) \textbf{do}
3: \quad Select \( u_k = u_j \in G \) s.t. \( j = \arg \max_{1 \leq l \leq K} d_k(u_l) \),
where \( d_k(u) = \min_{1 \leq m \leq k-1} (d(u, \bar{u}_m)) \)
4: \textbf{end for}
5: For current set of means \( \{\bar{u}_1, \ldots, \bar{u}_q\} \), assign each sam-
ple of \( G \) to the closest mean. That is,
\[ C(l) = \arg \min_{1 \leq k \leq q} ||u_l - \bar{u}_k||, \quad l = 1, \ldots, K \]
6: Given the updated cluster assignment \( C \), recompute the cluster
means \( \{\bar{u}_1, \ldots, \bar{u}_q\} \).
7: Steps 5 and 6 are iterated until either maximum iteration
has reached or the assignments do not change.
8: Reassign each samples of \( G \) to the closest fuzzy
antecedent, i.e.,
\[ C(l) = \arg \min_{1 \leq k \leq q} ||A(x_l)k||, \quad l = 1, \ldots, K \]
where \( A(.) \) will be defined in Eq. (15) of Section .
\end{algorithm}

\[ 1 \]For simplicity, we assume the covariance matrix \( \Sigma_x \) of a vari-
ate \( x \) is equal to \( \sigma_x^2 I \), where \( I \) denotes an identical matrix. Thus,
its corresponding scatter is \( \sigma_x = \sqrt{\Sigma_x} = \sqrt{d \sigma_x} \), where \( d \) denotes
the dimension of the variate \( x \).
the Gram-Schmidt procedure to \( M \) yields the desired basis vectors \( \mathbf{v}_l, l = 1, \ldots, q \).

Let \( V = [\mathbf{v}_1 \ldots \mathbf{v}_L] \). Thus the projected vector (feature) \( \mathbf{z} \) for the node \( G \) is

\[
\mathbf{z} = \mathbf{Vx}
\] (14)

**Estimating the Antecedent**

Each node (cluster) in the tree represents a rule antecedent. Without losing generality, we consider an internal node \( A_{k_1} \) and the goal is to estimate the membership function of \( A_{k_1k_2}, k_2 = 1, \ldots, q \). The other lower nodes can be handled similarly.

For notation convenience, we drop the subscript \( k_1 \) without causing ambiguity. We assume the antecedent \( A \) whose fuzzy membership function takes the form, for \( k_2 = 1, \ldots, q \)

\[
A_{k_2}(\mathbf{z}) = \exp[-(\mathbf{z} - \bar{\mathbf{z}}_{k_2})^T \mathbf{D}_{k_2}^{-1}(\mathbf{z} - \bar{\mathbf{z}}_{k_2})]
\] (15)

where \( \mathbf{z} \) is the projected vector on the MDF subspace; \( \mathbf{D}_{k_2} \) is a positive determine matrix and \( \bar{\mathbf{z}}_{k_2} \) is the center of the cluster, both needing to be estimated from the data. It is worthy noting that \( A_{k_2} \) is defined on the MDF subspace \( H \) rather than the original input space.

Let \( \mathbf{Z}_{k_2} = \begin{bmatrix} \mathbf{z}_1^T \ldots \mathbf{z}_{n_{k_2}}^T \end{bmatrix}^T \) be projected data matrix belonging to the cluster \( A_{k_2} \). We then write the equations to compute \( \mathbf{z} \) and \( D \) as follow

\[
\hat{\mathbf{z}}_{k_2} = \mathbf{Z}_{k_2} \mathbf{1}/n_{k_2}
\] (16)

and

\[
\hat{\mathbf{D}}_{k_2} = \alpha \hat{\Sigma}_{k_2} + \beta \overline{\Sigma} + \gamma \mathbf{I} \rho^2
\] (17)

where \( \hat{\Sigma}_{k_2} \) denotes each individual covariance matrix; \( \overline{\Sigma} \) is the common or within-class covariance matrix; \( \rho^2 \) is the shrunken scalar variance. Here weights \( \alpha, \beta \) and \( \gamma \) sums to 1 and allows a continuum of transition among quadratic discriminant (QDA), linear discriminant (LDA) and scalar covariance models. This method is very similar in flavor to (Friedman 1989; Hwang & Weng 2000) whose key point is regularization. By adjusting those weight it is possible to control the model complexity. QDA is the most complex model and scalar variance model is the less complex one. In practice, we set those parameters as a function of the number of samples belonging to the cluster.

**Summary of the Algorithms**

Algorithm 2 selects the subset of rules being fired by the input query \( \mathbf{x} \). Only rules with the \( q \) largest antecedent fuzzy measurement in the base are selected. Instead of exhaustive linear search the whole base, we use the following greedy heuristic tree search algorithm.

**Inference** The goal of inference is that given a base \( B \) and an input vector \( \mathbf{x} \), return the corresponding estimated output \( \hat{y} \). This involves the following steps:

which is huge for a large \( d \). Thus estimating \( W \) tends to overfit the data. Moreover, computing SVD of \( M \) is computationally expensive.

**Algorithm 2** Select rules: Given a rule base \( R \) and an input vector \( \mathbf{x} \), return the \( q \) rules whose antecedent clusters are the closest to the query input.

1: \( c, p \leftarrow \) the root node of \( R \).
2: for \( c \) has been split into sub-clusters do
3: \( p \leftarrow c \)
4: \( c \leftarrow \) the \( m \)th child of the node \( c \), where \( m = \arg\min \{x_k \in \mathbb{R}^q | A_{k_2}(\mathbf{x}) \} \)
5: end for
6: Return \( p \) and its child nodes which correspond to \( q \) rules.

- use Algorithm 2 finding the parent node \( p \) and its associated clusters \( \{A_i \mid i = 1, \ldots, q \} \)
- compute the projected feature vector \( \mathbf{z} = \mathbf{Vx} \), where \( V \) denotes the projection matrix of the MDF space derived by the node \( p \)
- compute the \( y_i = \theta_i^T \mathbf{z} \) and \( \bar{y} = \frac{\sum_{i=1}^{q} \mathbf{A}_i(x)}{\sum_{i=1}^{q} \mathbf{A}_i(x)} \) (see Eqs. (5) and (4) respectively)

**Learning** The learning of the proposed fuzzy model contains two phases: structure identification and parameter identification.

In structure identification, a set of initial hierarchical fuzzy rules are discovered from given a collected labeled sample \( G = \{ (\mathbf{x}_l, y_l) \mid l = 1, \ldots, K \} \). The result is the hierarchical organized rule base \( B \). This involves

- recursively split \( G \) by using Algorithm 1 until all clusters either the number of members is less than a predefined value \( N_T \) or the dissimilarity of all members of each one from one another is less than a threshold \( d_T \)
- estimate parameters \( \theta \) of Eq. (5) by performing the weighted-least-square on each leaf node using its associated data samples (see Eq. (6)).

In parameter identification, we assume the structure of the rule base is fixed. Unlike the structure learning, this step can be done incrementally, namely, the proposed model is capable of adapting based on new coming data pair \( (\mathbf{x}_l, y_l) \). This involves

- as in inference, use Algorithm 2 finding the parent node \( p \) and its associated clusters \( \{A_i \mid i = 1, \ldots, q \} \) and compute the projected vector onto the subspace derived by \( p \), i.e., \( \mathbf{z} = \mathbf{Vx} \)

- for all \( i = 1, \ldots, q \) apply the sequence sequence least square equations listed in Eq. (10) to update the parameters in the consequent part of the rules

**Experiments**

**Simulation**

We show experimentally the proposed method’s feature extraction and real-time learning capabilities on an artificial dataset. As a first test (2D-cross), we ran the method on noisy training data drawn from a two dimensional function

\[
y = \max\{\exp(-10x_1^2), \exp(-50x_2^2), 1.25\exp(-5(x_1^2 + x_2^2)) + N(0, 0.1^2)\}
\]
as shown in Fig. 2 (a). This function is a combination of nonlinear and linear areas and an interesting test of learning and generalization capabilities of a learning algorithm (Vijayakumar & Schaal 2000): learning system with simple model find it hard to capture the nonlinearity well, while more complex models easily overfit the linear areas. A second test (100d-cross) added 98 irrelevant noise features to the input, each having a density $N(0, 0.025^2)$. We thus obtain a 100-dimension input space. A third test (200d-cross) added another 100 irrelevant noise features with the density $N(0, 0.05)$ to the input space of the second test. The learning curves with these data set are illustrated in Fig. 2 (c). In all three cases, the normalized mean square error (nMSE) is reported on an independent test set (1681 points on a $41 \times 41$ grid in the unit-square in the input space). As the number of training number increasing, all nMSEs converged to the nice function approximation result whose $nMSE < 0.03$ after 100,000 training data points. Fig. 2 (b) shows the learned surface of the third test after 100,000 training samples presented. Fig. 2 (d) illustrates the execution time of both training and test processes with respect the number of training samples. It is interesting to note that the execution time increases linearly w.r.t. the number of training samples. The reason is that the learning and inference procedures have the logarithmic complexity and, thus, the average time of adding a training sample and retrieving a testing sample does not change much even though the size of tree has grow tremendously. As considering the third case (200d-cross), execution time on a 200-dimension data set takes only about 500 seconds for 100,000 sample, in other words, the average time for a sample is about 5 milliseconds. This is fast and is extremely important for later real-time robot navigation experiment.

In summary, the power of feature extraction is due to finding the MDF subspace. The real-time learning performance is achieved by the hierarchical organization of the rule base.

Range-based Wall Following Behavior

Our mobile humanoid robot, Dav (Zeng, Cherba, & Weng 2003), was used to test our proposed framework. In this experiment, we trained the robot to navigate along the loop shown in Fig. 3. The goal is to learn the mapping from a laser ranger’s raw reading $x$ (dim($x$) = 361) to the robot’s heading and speed $y$.

We identified five typical types of corridor in this test site:
The power of the proposed method is demonstrated through its ability to model complex, high-dimensional input-output functions. This is achieved through a locally model-fitting approach that partitions the input space into regions and fits a simple model to the data within each region. The success of this method is evident in the low error rate of the leave-one-out test, which is shown in Fig. 5 (a). The average error histogram of the five tests is shown in Fig. 5 (b), with a plot of the predicted outputs on the x-axis and the actual outputs on the y-axis. The error rate of about 50% indicates that the proposed method is effective in handling high-dimensional input.


desired behavior $y$. Such a function is typically so complex that writing a program to simulate it accurately is not possible. The success of learning for such a high-dimensional input is mainly due to the discriminating feature extraction, and the real-time speed is due to the logarithmic time complexity.

The key idea of this paper is locally model-fitting whose learning algorithm first partitions the space into local regions. In each of those regions, a simple model (e.g., TS model) is used to model the input-output function. The number and the organization of local models account for the nonlinearity and the complexity of the problem. This method is suitable for incremental learning, especially in the situation where limited knowledge exists about the target system with high-dimensional input.

**References**


**Conclusion**

This paper describes a learning fuzzy framework to model system dynamics. The power of the proposed method is to enable the machine to learn a very complex function $y = f(x)$ between the input (e.g., sensory input $x$) and the output desired behavior $y$. Such a function is typically so complex that writing a program to simulate it accurately is not possible. The success of learning for such a high-dimensional input is mainly due to the discriminating feature extraction, and the real-time speed is due to the logarithmic time complexity.