A Fuzzy Elman Neural Network

Ling Li, Zhidong Deng, and Bo Zhang

The State Key Lab of Intelligent Technology and Systems
Dept. of Computer Science, Tsinghua University
Beijing 100084, China

Abstract — A fuzzy Elman neural network (FENN) is proposed to identify and simulate nonlinear dynamic systems. Each of all the fuzzy rules used in FENN has a linear state-space equation as its consequence and the network, by use of firing strengths of input variables, combines these Takagi-Sugeno type rules to represent the modeled nonlinear system. The context nodes in FENN are used to perform temporal recurrence. An online dynamic BP-like learning algorithm is derived. The pendulum system is simulated as a testbed for illustrating the better learning and generalization capability of the proposed FENN network, compared with the common Elman-type networks.

Keywords — nonlinear dynamic system modeling, fuzzy neural networks, Elman networks, BP-like learning algorithm.

1. INTRODUCTION

Artificial neural networks (ANNs), including fuzzy neural networks (FNNs), are essentially nonlinear. They have already been used to identify, simulate and control nonlinear systems [1,2] and have been proved to be universal approximators [3,4,5]. As compared with ANNs, FNNs can merge human experience into the networks through designating some rules based on prior knowledge. These fuzzy rules in the trained network are also easy to understand.

Recurrent networks, especially the Elman networks [6], are often adopted to identify or generate the temporal outputs of nonlinear systems. It is well known that a recurrent network is capable of approximating a finite state machine [7] and thus can simulate any time series. So recurrent networks are now widely used in fields concerned with temporal problems. In published literature, however, all the initial weights of recurrent networks are set randomly instead of using any prior knowledge and thus the trained networks are vague to human and their convergence speed is slow. In addition, the temporal generalization capability of simple recurrent networks is not so good [8]. These two major problems make the applications of recurrent networks with temporal identification and control of systems more difficult.

In this paper, a novel network structure called FENN (Fuzzy Elman Neural Network) is proposed. It is motivated for integrating fuzzy neural networks with the Elman networks so that the above two problems are addressed to a certain degree. This integrated network uses the combination of linear state-space equations as its rule consequence with firing strengths of input variables to express a nonlinear dynamic system. Due to the fact that the context nodes in FENN are conceptually taken from the Elman networks, FENN is also a

dynamic network and can be used for reproducing temporal trajectories of the modeled system. Starting from either some prior knowledge or zero-knowledge (random initial weight settings), FENN can be trained from one or more temporal trajectories of the modeled nonlinear system by using a dynamic BP-like learning algorithm. Thus, knowledge can be put into the network *a priori* and extracted easily after the network is trained. The simulation results obtained in this paper illustrate the superior performance of the proposed dynamic network.

This paper is organized as follows. In Section 2, the network structure of FENN is proposed. The corresponding learning algorithm is described in detail in Section 3. Section 4 takes a numerical example for demonstrating the feasibility of the proposed FENN. In the last section, conclusions are drawn and some future works are discussed.

2. NETWORK STRUCTURE

In this section, we introduce our method to describe a nonlinear system by using fuzzy rules in the form of linear state-space equations as consequences. The Takagi-Sugeno type fuzzy rules are discussed in detail in Subsection A. In Subsection B, the network structure of FENN is presented.

A. Fuzzy rules

Recently, more and more attention has paid to the Takagi-Sugeno type rules [9] in studies of fuzzy neural networks. This significant inference rule provides an analytic way of analyzing the stability of fuzzy control systems. If we combine the Takagi-Sugeno controllers together with the controlled system and use state-space equations to describe the whole system [10], we can get another type of rules to describe nonlinear systems as below:

Rule r: IF x_1 is $T_{x_1}^r$ AND ... AND x_N is $T_{x_N}^r$ AND
u_1 is $T_{u_1}^r$ AND ... AND u_M is $T_{u_M}^r$
THEN $\dot{\boldsymbol{X}} = \boldsymbol{A}^r \boldsymbol{X} + \boldsymbol{B}^r \boldsymbol{U}$

where $\boldsymbol{X} = \begin{bmatrix} x_1 & x_2 & \dots & x_N \end{bmatrix}^T$ is the inner state vector of the nonlinear system, $\boldsymbol{U} = \begin{bmatrix} u_1 & u_2 & \dots & u_M \end{bmatrix}^T$ is the input vector to the system, and *N*, *M* are the dimensions; $T_{x_i}^r$ and $T_{u_j}^r$ are linguistic terms (fuzzy sets) defining the conditions for x_i and u_j respectively, according to Rule r; $\boldsymbol{A}^r = (a_{ij}^r)_{N\times N}$ is a matrix of $N \times N$ and $\boldsymbol{B}^r = (b_{ij}^r)_{N\times M}$ of $N \times M$.

Though induced from the Takagi-Sugeno type rules and the controlled system, the above form of rules are suitable to simulate or identify any nonlinear systems, whether with or without controllers. The antecedent of one such rule defines a fuzzy subspace of $\boldsymbol{X}$ and $\boldsymbol{U}$, and the consequence tells which linear system can the nonlinear system be regarded as in that subspace.

When considered in discrete time, such as modeling using a digital computer, we often use the discrete state-space equations instead of the continuous version. Concretely, the fuzzy rules become:

Rule r: IF $x_1(t)$ is $T_{x_1}^r$ AND ... AND $x_N(t)$ is $T_{x_N}^r$ AND
$u_1(t)$ is $T_{u_1}^r$ AND ... AND $u_M(t)$ is $T_{u_M}^r$
THEN $\boldsymbol{X}(t+1) = \boldsymbol{A}^r \boldsymbol{X}(t) + \boldsymbol{B}^r \boldsymbol{U}(t)$

where $\boldsymbol{X}(t) = \left[x_1(t) \quad x_2(t) \quad \dots \quad x_N(t)\right]^T$ is the discrete sample of state vector at discrete time *t*. In following discussion we shall use the latter form of rules.

In both forms, the output of the system is always defined as:

$$\boldsymbol{Y} = \boldsymbol{C}\boldsymbol{X} \quad (\text{or } \boldsymbol{Y}(t) = \boldsymbol{C}\boldsymbol{X}(t)), \tag{1}$$

where $\boldsymbol{C} = (c_{ij})_{P\times N}$ is a matrix of $P \times N$, and *P* is the dimension of output vector $\boldsymbol{Y}$.

The fuzzy inference procedure is specified as below. First, we use multiplication as operation AND to get the firing strength of Rule r:

$$f_r = \prod_{i=1}^{N} \mu_{T_{x_i}^r}\left[x_i(t)\right] \cdot \prod_{j=1}^{M} \mu_{T_{u_j}^r}\left[u_j(t)\right], \tag{2}$$

where $\mu_{T_{x_i}^r}$ and $\mu_{T_{u_j}^r}$ are the membership functions of $T_{x_i}^r$ and $T_{u_j}^r$, respectively. After normalization of the firing strengths, we get (assuming *R* is the total number of rules)

$$S = \sum_{r=1}^{R} f_r, \quad h_r = f_r / S, \tag{3}$$

where *S* is the summation of firing strengths of all the rules, and h_r is the normalized firing strength of Rule r. When the defuzzification is employed, we have

$$\begin{aligned} \boldsymbol{X}^r(t+1) &= \boldsymbol{A}^r \boldsymbol{X}(t) + \boldsymbol{B}^r \boldsymbol{U}(t), \\ \boldsymbol{X}(t+1) &= \sum_{r=1}^{R} h_r \boldsymbol{X}^r(t+1) = \sum_{r=1}^{R} h_r \left[\boldsymbol{A}^r \boldsymbol{X}(t) + \boldsymbol{B}^r \boldsymbol{U}(t)\right] \\ &= \left(\sum_{r=1}^{R} h_r \boldsymbol{A}^r\right)\boldsymbol{X}(t) + \left(\sum_{r=1}^{R} h_r \boldsymbol{B}^r\right)\boldsymbol{U}(t) \\ &= \boldsymbol{A}\boldsymbol{X}(t) + \boldsymbol{B}\boldsymbol{U}(t), \end{aligned} \tag{4}$$

where

$$\boldsymbol{A} = \sum_{r=1}^{R} h_r \boldsymbol{A}^r, \boldsymbol{B} = \sum_{r=1}^{R} h_r \boldsymbol{B}^r \ . \tag{5}$$

Using equation (4), the system state transient equation, we can calculate the next state of system by current state and input.

B. Network structure

Figure 1 shows the seven-layer network structure of FENN, with the basic concepts taken from the Elman networks and fuzzy neural networks. In this network, input nodes which accept the environment inputs and context nodes which copy the value of the state-space vector from layer 5 are all at layer 1 (the Input Layer). They represent the linguistic variables known as u_j and x_i in the fuzzy rules. Nodes at layer 2 act as the membership functions, translating the linguistic variables from layer 1 into their membership degrees. Since there may exist several terms for one linguistic variable, one node in layer 1 may have links to several nodes in layer 2, which is accordingly named as the term nodes. The number of nodes in the Rule Layer (layer 3) and the one of the fuzzy rules are the same - each node represents one fuzzy rule and calculates the firing strength of the rule using

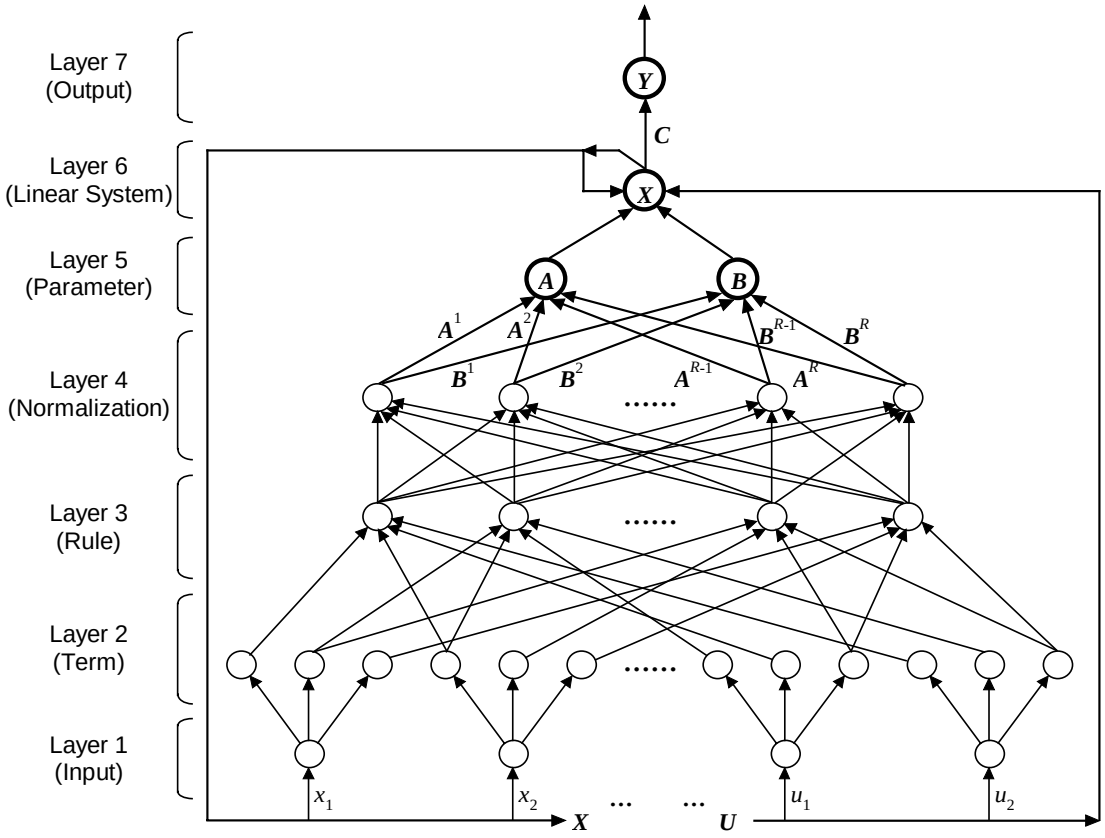

Figure 1 The seven-layer structure of FENN

membership degrees from layer 2. The connections between layer 2 and layer 3 correspond with the antecedent of each fuzzy rule. Layer 4, as the Normalization Layer, simply does the normalization of the firing strengths. Then with the normalized firing strengths h_r, rules are combined at layer 5, the Parameter Layer, where $\boldsymbol{A}$ and $\boldsymbol{B}$ become available. In the Linear System Layer, the 6th layer, current state vector $\boldsymbol{X}(t)$ and input vector $\boldsymbol{U}(t)$ are used to get the next state $\boldsymbol{X}(t+1)$, which is also fed back to the context nodes for fuzzy inference at time $(t+1)$. The last layer is the Output Layer, multiplying $\boldsymbol{X}(t+1)$ with $\boldsymbol{C}$ to get $\boldsymbol{Y}(t+1)$ and outputting it.

Next we shall describe the feedforward procedure of FENN by giving the detailed node functions of each layer, taking one node per layer as example. We shall use notations like $u_i^{[k]}$ to denote the *i*th input to the node in layer k, and $o^{[k]}$ the output of the node in layer k. Another issue to mention here is the initial values of the context nodes. Since FENN is a recurrent network, the initial values are essential to the temporal output of the network. Usually they are preset to 0, as zero-state, but non-zero initial state is also needed for some particular case.

Layer 1: each node in this layer has only one input, either from the environment or the Parameter Layer. Function of nodes is to transmit the input values to the next layer, i.e.,

$$o^{[1]} = u^{[1]}.$$

Layer 2: there is only one input to each node at layer 2. That is, each term node can link to only one node at layer 1, though each node at layer 1 can link to several nodes at layer 2 (as described before). The Gaussian function is adopted here as the membership function:

$$o^{[2]} = e^{-\frac{(u^{[2]}-c^r)^2}{2(s^r)^2}}, \tag{6}$$

where c^r and s^r give the center (mean) and width (variation) of the corresponding linguistic term of input $u^{[2]}$ in Rule r, i.e., one of $T^r_{x_i}$ or $T^r_{u_j}$.

Layer 3: in the Rule Layer, the firing strength of each rule is determined [see (2)]. Each node in this layer represents a rule and accepts the outputs of all the term nodes associated with the rule as inputs. The function of node is fuzzy operator AND: (multiplication here)

$$o^{[3]} = \prod_i u_i^{[3]}. \tag{7}$$

Layer 4: the Normalization Layer also has the same number of nodes as the rules, and is fully connected with the Rule Layer. Nodes here do the function of (3), i.e.,

$$o^{[4]} = u^{[4]} \Big/ \sum_i u_i^{[4]}. \tag{8}$$

In (8) we use $u^{[4]}$ to denote the specific input corresponding to the same rule with the node.

Layer 5: this layer has two nodes, one for figuring matrix $\boldsymbol{A}$ and the other for $\boldsymbol{B}$. Though we can use many nodes to represent the components of $\boldsymbol{A}$ and $\boldsymbol{B}$ separately, it is more convenient to use matrices. So with a little specialty, its weights of links from layer 4 are matrices $\boldsymbol{A}^r$ (to node for $\boldsymbol{A}$) and $\boldsymbol{B}^r$ (to node for $\boldsymbol{B}$). It is also fully connected with the previous layer. The functions of nodes for $\boldsymbol{A}$ and $\boldsymbol{B}$ are

$$o^{[5]}_{\text{for } \boldsymbol{A}} = \sum_{r=1}^{R} u_r^{[5]} \boldsymbol{A}^r, o^{[5]}_{\text{for } \boldsymbol{B}} = \sum_{r=1}^{R} u_r^{[5]} \boldsymbol{B}^r \tag{9}$$

respectively.

Layer 6: the Linear System Layer has only one node, which has all the outputs of layer 1 and layer 5 connected to it as inputs. Using matrix form of inputs and output, we have [see (5)]

$$o^{[6]} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}\boldsymbol{U} = o^{[5]}_{\text{for } \boldsymbol{A}} o^{[1]}_{\text{context}} + o^{[5]}_{\text{for } \boldsymbol{B}} o^{[1]}_{\text{input}}.$$

So the output of layer 6 is $\boldsymbol{X}(t+1)$ in (4).

Layer 7: simply as layer 1, the unique node in the Output Layer passes the input value from layer 6 to output. The only difference is that the weight of the link is matrix $\boldsymbol{C}$, not unity,

$$\boldsymbol{Y} = o^{[7]} = \boldsymbol{C} \cdot u^{[7]} \tag{10}$$

This proposed network structure implements the dynamic system combined by our discrete fuzzy rules and the structure of recurrent networks. With preset human knowledge, the network can do some tasks well. But it will do much better after learning rules from teaching examples. In the next section, a learning algorithm will be put forth to adjust the variable parameters in FENN, such as c^r, s^r, $\boldsymbol{A}^r$, $\boldsymbol{B}^r$, and $\boldsymbol{C}$.

3. LEARNING ALGORITHM

Learning of the parameters is based on sample temporal trajectories. In this section, a learning algorithm which learns a single trajectory per iteration by points (STP, Single Trajectory learning by Points) will be proposed.

In the STP learning algorithm, one iteration is comprised of all the time points of the

learning trajectory, and the network parameters are updated online. At one time point, FENN uses the current value of parameters to get the output, and runs the learning algorithm to adjust the parameters. Then in the next time point, the updated parameters are used, and learning will be processed again. After the whole trajectory was passed, one iteration completes and in the next iteration, the same trajectory or an other one would be learned.

Given the initial state $\boldsymbol{X}(0)$ and the desired output $\boldsymbol{Y}_d(t),\ t=1,2,\ldots,t_e$, the error at time *t* is defined as

$$e(t)=\frac{1}{2}\left\|\boldsymbol{Y}_d(t)-\boldsymbol{Y}(t)\right\|^2=\frac{1}{2}\sum_{i=1}^{P}\left[y_{di}(t)-y_i(t)\right]^2, \tag{11}$$

and the target of learning is to minimize each $e(t)$, t=1,2,…,t_e. The gradient descent technique is used here as a general learning rule: (assuming *w* is an adjustable parameter, e.g. a_{ij}^r)

$$\Delta w(t)\propto\frac{\partial e(t)}{\partial w(t)},$$

$$w(t+1)=w(t)+\Delta w(t)=w(t)-\eta\frac{\partial e(t)}{\partial w(t)}, \tag{12}$$

where $\eta>0$ is the learning rate. We shall show how to compute $\partial e(t)/\partial w(t)$ in a recurrent situation, giving both the equations in a general case and for specified parameters. If possible, we shall also give the matrix form of the equations, for its concision and efficiency.

From (1) and (11) we can get

$$\frac{\partial e(t)}{\partial x_i(t)}=\sum_{j=1}^{P}\frac{\partial e(t)}{\partial y_j(t)}\frac{\partial y_j(t)}{\partial x_i(t)}=-\sum_{j=1}^{P}\left[y_{dj}(t)-y_j(t)\right]c_{ji}(t),$$

or in matrix form

$$\frac{\partial e(t)}{\partial \boldsymbol{X}^T(t)}=-\left[\boldsymbol{Y}_d(t)-\boldsymbol{Y}(t)\right]^T\boldsymbol{C}(t).$$

Since we want to compute $\partial e(t)/\partial w(t)$, we should also know the derivative of $\boldsymbol{X}(t)$ to the adjustable parameter *w*. Taking into account the recurrent property [see (4)], we have

$$\frac{\partial^+ x_i(t)}{\partial w}=\frac{\partial x_i(t)}{\partial w}+\sum_{j=1}^{N}\frac{\partial x_i(t)}{\partial x_j(t-1)}\frac{\partial^+ x_j(t-1)}{\partial w}$$

or in matrix form,

$$\frac{\partial^+ \boldsymbol{X}(t)}{\partial w}=\frac{\partial \boldsymbol{X}(t)}{\partial w}+\frac{\partial \boldsymbol{X}(t)}{\partial \boldsymbol{X}^T(t-1)}\frac{\partial^+ \boldsymbol{X}(t-1)}{\partial w}, \tag{13}$$

which is a recursive definition of the ordered derivative $\partial^+\boldsymbol{X}(t)/\partial w$. With the initial value ($\partial^+\boldsymbol{X}(0)/\partial w$) given, we can calculate $\partial^+\boldsymbol{X}(t)/\partial w$ step by step, and use

$$\frac{\partial e(t)}{\partial w}=\frac{\partial e(t)}{\partial \boldsymbol{X}^T(t)}\frac{\partial^+ \boldsymbol{X}(t)}{\partial w}=-\left[\boldsymbol{Y}_d(t)-\boldsymbol{Y}(t)\right]^T\boldsymbol{C}(t)\frac{\partial^+ \boldsymbol{X}(t)}{\partial w} \tag{14}$$

and (12) to update *w*.

From (4) and (5) we can get

$$\frac{\partial x_k(t)}{\partial a_{ij}^r(t)} = \delta_{ki} h_r x_j(t-1),$$
$$\frac{\partial x_k(t)}{\partial b_{ij}^r(t)} = \delta_{ki} h_r u_j(t-1), \tag{15}$$

where δ_{ki} is the Kronecker symbol which is 1 when *k* and *i* are equal, otherwise 0. Together with (3), we have

$$\frac{\partial x_k(t)}{\partial f_r} = \frac{x_k^r(t) - x_k(t)}{S},$$

Since [see (2) and (6)]

$$\frac{\partial f_r}{\partial x_j(t-1)} = \frac{\partial f_r}{\partial \mu_{T_{x_j}^r}} \frac{\partial \mu_{T_{x_j}^r}}{\partial x_j(t-1)} = -f_r \frac{x_j(t-1) - c_{T_{x_j}^r}}{s_{T_{x_j}^r}^2},$$

we can get

$$\begin{aligned} \frac{\partial x_i(t)}{\partial x_j(t-1)} &= a_{ij} + \sum_{r=1}^{R} \frac{\partial x_i(t)}{\partial f_r} \frac{\partial f_r}{\partial x_j(t-1)} \\ &= a_{ij} - \sum_{r=1}^{R} h_r \left[x_i^r(t) - x_i(t) \right] \frac{x_j(t-1) - c_{T_{x_j}^r}}{s_{T_{x_j}^r}^2}. \end{aligned} \tag{16}$$

Using (13), (15) and (16), we can calculate the ordered derivative for a_{ij}^r and b_{ij}^r.

Though we can easily get equations below from (2) and (6),

$$\begin{aligned} \frac{\partial f_r}{\partial c_{T_{x_i}^r}} &= f_r \frac{x_i(t-1) - c_{T_{x_i}^r}}{s_{T_{x_i}^r}^2}, \quad \frac{\partial f_r}{\partial s_{T_{x_i}^r}} = f_r \frac{[x_i(t-1) - c_{T_{x_i}^r}]^2}{s_{T_{x_i}^r}^3}, \\ \frac{\partial f_r}{\partial c_{T_{u_j}^r}} &= f_r \frac{u_j(t-1) - c_{T_{u_j}^r}}{s_{T_{u_j}^r}^2}, \quad \frac{\partial f_r}{\partial s_{T_{u_j}^r}} = f_r \frac{[u_j(t-1) - c_{T_{u_j}^r}]^2}{s_{T_{u_j}^r}^3}, \end{aligned} \tag{17}$$

the derivatives to the parameters of membership functions, i.e., *c* and *s*, are not so easy to get in that there exists the probability of two or more rules using the same linguistic term. If we assign each linguistic term a different serial number, said v, from 1 to V, then the linguistic term T_v may be used in Rule r_1, r_2, … That is, it may be called $T_{x_i}^{r_1}$ (or $T_{u_i}^{r_1}$), $T_{x_j}^{r_2}$ (or $T_{u_j}^{r_2}$), …, in the previous part of this paper. To clearly note this point, we shall use the notations c_v and s_v to represent the center and the width of the membership function μ_v of term T_v, and $\hat{x}_v^r$ the corresponding input variable with T_v in Rule r, no matter it is x_i or u_j. Thus (17) becomes

$$\frac{\partial f_r}{\partial c_v} = f_r \frac{\hat{x}_v^r(t-1) - c_v}{s_v^2}, \quad \frac{\partial f_r}{\partial s_v} = f_r \frac{\left[\hat{x}_v^r(t-1) - c_v \right]^2}{s_v^3},$$

and we can calculate $\partial x_i(t)/\partial c_v$ and $\partial x_i(t)/\partial s_v$ as

$$\begin{aligned} \frac{\partial x_i(t)}{\partial c_v} &= \sum_r \frac{\partial x_i(t)}{\partial f_r} \frac{\partial f_r}{\partial c_v} = \sum_r h_r \left[x_i^r(t) - x_i(t) \right] \frac{\hat{x}_v^r(t-1) - c_v}{s_v^2}, \\ \frac{\partial x_i(t)}{\partial s_v} &= \sum_r \frac{\partial x_i(t)}{\partial f_r} \frac{\partial f_r}{\partial s_v} = \sum_r h_r \left[x_i^r(t) - x_i(t) \right] \frac{\left[\hat{x}_v^r(t-1) - c_v \right]^2}{s_v^3}, \end{aligned} \tag{18}$$

where summation is for all the rules containing T_v. So, using (13) with (16) and (18), $\partial^+ \boldsymbol{X}(t)/\partial c_v$ and $\partial^+ \boldsymbol{X}(t)/\partial s_v$ are available.

The updating of matrix $\boldsymbol{C}$ is really simple and plain. By (1) or (10), we have

$$\frac{\partial e(t)}{\partial c_{ij}(t)} = \frac{\partial e(t)}{\partial y_i(t)} \frac{\partial y_i(t)}{\partial c_{ij}(t)} = -\left[y_{di}(t) - y_i(t)\right] x_j(t),$$

or in matrix form

$$\frac{\partial e(t)}{\partial \boldsymbol{C}(t)} = -\left[\boldsymbol{Y}_d(t) - \boldsymbol{Y}(t)\right] \boldsymbol{X}^T(t),$$

and from (12), $\boldsymbol{C}$ is updated by

$$\begin{aligned} c_{ij}(t+|1) &= c_{ij}(t) + \eta\left[y_{di}(t) - y_i(t)\right] x_j(t) \\ \boldsymbol{C}(t+1) &= \boldsymbol{C}(t) + \eta\left[\boldsymbol{Y}_d(t) - \boldsymbol{Y}(t)\right] \boldsymbol{X}^T(t) \end{aligned} . \tag{19}$$

With (12~16) and (18~19), all the updating equations are given. Some of them are recursive, reflecting the recurrent property of FENN. The initial values of those recurrent items, such as $\partial^+ x_k(t)/\partial a_{ij}^r$ in (15), are set to zero in the beginning of learning. Because of the gradient descent characteristics, our STP learning algorithm is also called a BP-like learning algorithm, or RTRL (real-time recurrent learning) as in [11].

When learning a nonlinear system, different trajectories are needed to overall describe the system. Usually, multiple trajectories are learned one by one, and one pass of such learning (called a cycle) is repeated until some *training convergence criterion* is met. A variety of such cycle strategy, which does not distribute the learning iterations among every trajectories evenly in one cycle, may produce more efficient learning. In such unevenly strategy, we can give more learning chances (iterations) to the less learned trajectory (often with larger error), and thus speed up the total learning. Next section we will show how to do this by an example.

4. COMPUTER SIMULATION - THE PENDULUM SYSTEM

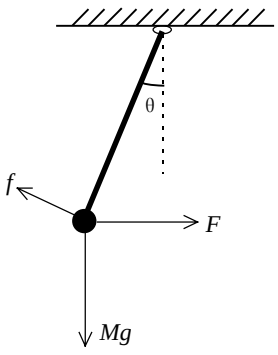

Figure 2 The pendulum system

We employ the pendulum system to test the capability and generalization of FENN. Figure 2 gives the scheme of the proposed system. A rigid zero-mass pole with length *L* connects a pendulum ball and a frictionless pivot at the ceiling. The mass of the pendulum ball is *M*, and its size can be omitted with respect to *L*. The pole (together with the ball) can rotate around the pivot, against the friction *f* from the air to the ball, which can be simply quantified as:

$$f = -\mathrm{sgn}(v) \cdot Kv^2, \tag{20}$$

where $v = L\dot{\theta}$ is the line velocity of the pendulum ball, and θ is the angle between the pole and the vertical direction. The item $-\mathrm{sgn}(v)$ (sgn(*v*) is the sign of *v*) in (20) shows that *f* always counteracts the movement of the ball, and its direction is perpendicular with the moving pole.

If we exert a horizontal force to the ball, or give the pendulum system a non-zero initial position ($\theta \neq 0$) or velocity ($\dot{\theta} \neq 0$), the ball will rotate around the pivot. Below is

its kinetic equation,

$$\ddot{\theta} = \frac{F}{ML}\cos\theta - \frac{g}{L}\sin\theta - \mathrm{sgn}(\dot{\theta})\frac{KL}{M}\dot{\theta}^2,$$

where $g = 9.8 m/s^2$ is the acceleration of gravity. Using two state variables x_1, x_2 to represent θ and $\dot{\theta}$ respectively, the state-space equation of the system is (for simplicity, let *K*, *L*, *M* all be 1)

$$\begin{aligned} \boldsymbol{X} &= \begin{pmatrix} x_1 & x_2 \end{pmatrix}^T = \begin{pmatrix} \theta & \dot{\theta} \end{pmatrix}^T, \boldsymbol{U} = (F), \\ \dot{\boldsymbol{X}} &= \begin{pmatrix} 0 & 1 \\ -g\sin x_1 / x_1 & -|x_2| \end{pmatrix} \boldsymbol{X} + \begin{pmatrix} 0 \\ \cos x_1 \end{pmatrix} \boldsymbol{U}. \end{aligned} \tag{21}$$

Applying 5-order Runge-Kutta method to (21), we can get the 'continuous' states of the testing system. The input (***U***) and states (***X***) are sampled every T $(=0.25)$ second and the total time is 25 second. Thus the number of sample points is $t_e = 25/T = 100$. Given initial state $\boldsymbol{X}(0)$, by sampling we can get $\boldsymbol{U}(0) \sim \boldsymbol{U}(t_e - 1)$ and $\boldsymbol{X}(1) \sim \boldsymbol{X}(t_e)$, where

$$\boldsymbol{U}(t) = \left(F[(t+\tfrac{1}{2})T]\right), \quad \boldsymbol{X}(t) = \begin{pmatrix} \theta(t \cdot T) & \dot{\theta}(t \cdot T) \end{pmatrix}^T.$$

In this way, we got 12 trajectories, with different combinations of force *F* and initial state ***X***. (see Table 1)

We use three linguistic terms for each state variable (see Table 3), which are *Negative*, *Zero* and *Positive*. (Though using the same name, the term *Positive* for x_1 is independent with the one for x_2, and so are *Zero* and *Negative*.) Thus there are totally nine rules, i.e., $R = 9$. Before training, we set all the $\boldsymbol{A}^r$ and $\boldsymbol{B}^r$ to zero, and ***C*** to unity, making the state-space vector ***X*** the output. We use the first t_L $(=20)$ data of trajectories 1~5 to train FENN, and test it with all the $t_e\,(=100)$ data of all the twelve trajectories. The strategy of learning multiple trajectories mentioned in last section is performed as: each learning cycle is made up by ten iterations, five of which are equally allotted to those learned trajectories while the rest five are scattered with the number proportional to the current error of the trajectories. Adaptive learning rate is also adopted in learning.

In the first stage of learning, only $\boldsymbol{A}^r$ and $\boldsymbol{B}^r$ are learned to set up the initial fuzzy rules, leaving the membership parameters and matrix ***C*** unmodified. After 1200 cycles of learning, we get a very impressive result, which is presented in Figure 4 to Figure 15. (The continuous curve and dashed curve indicate the desired curves of θ and $\dot{\theta}$, respectively; the notation 'o' and '+' represent the actual discrete outputs of FENN.) To diminish the space of figures, only the first 50 data points of each trajectory (except trajectory 12) are shown, with the RMS errors of state x_1 (θ) and x_2 $(\dot{\theta})$ listed respectively above the

Table 1 Twelve data sets of the pendulum system (the variable *t* in the table is a continuous parameter, not the discrete *t* in FENN.)

No.	Initial State ***X***	Force *F*	No.	Initial State ***X***	Force *F*
1	$(\pi/2, 0)^T$	0	7	$(5\pi/12, -1)^T$	$4\sin(2t)$
2	$(\pi/2, -1)^T$	$4\sin(2t)$	8	$(-\pi/2, 0)^T$	0
3	$(-\pi/2, 1)^T$	$4\sin(2t)$	9	$(5\pi/12, -1)^T$	$6\sin(2t)$
4	$(0, 2.5)^T$	$8\sin(2t)$	10	$(5\pi/12, 0)^T$	$5\sin(5t)$
5	$(0, 0)^T$	see Figure 3	11	$(5\pi/12, 0)^T$	see Figure 3
6	$(5\pi/12, 0)^T$	0	12	$(0, 0)^T$	see Figure 3

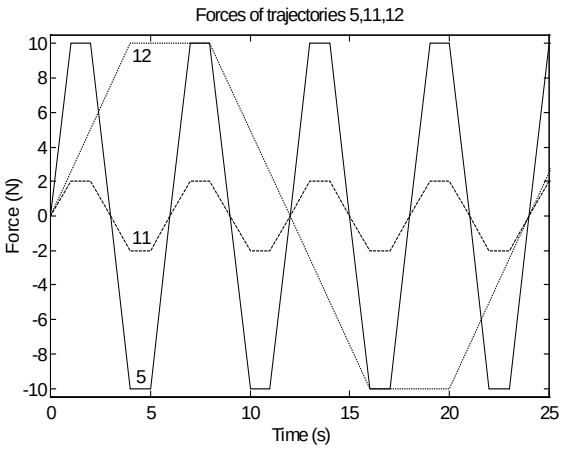

Figure 3 Forces of trajectories 5,11,12

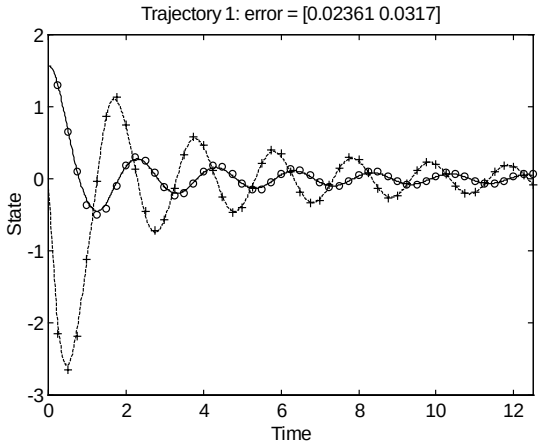

Figure 4

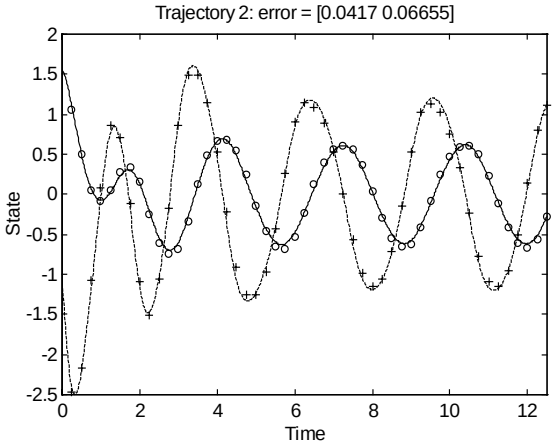

Figure 5

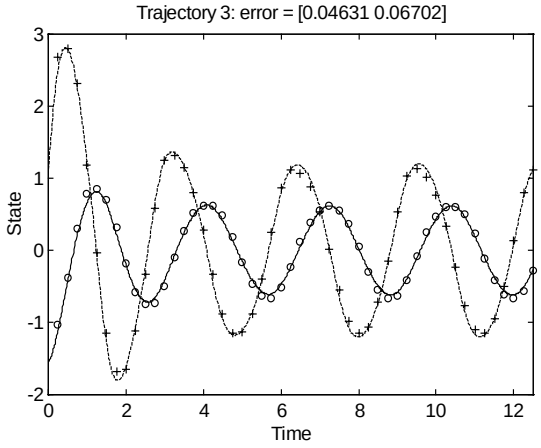

Figure 6

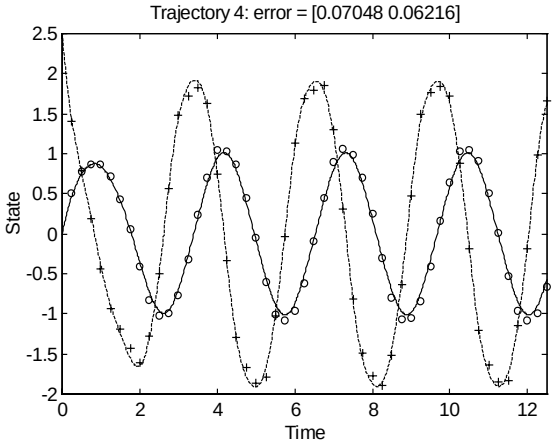

Figure 7

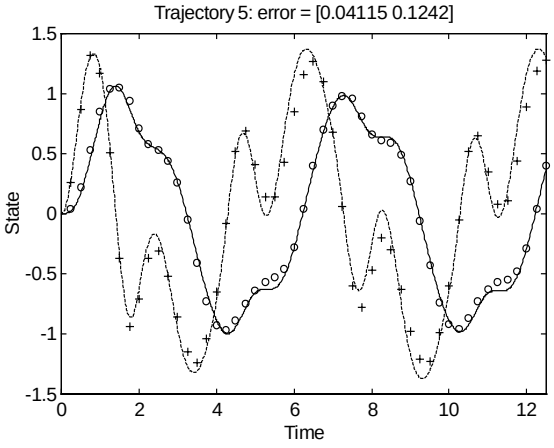

Figure 8

figures.

Though trained with only t_L $(=20)$ data points of the first five trajectories, FENN exhibits a great capability of generalization and succeeds in simulating the pendulum system at data points of $t > t_L$ and/or under some unlearned conditions. Following are discussions about the simulation result:

- ✧ Among the learned trajectories, the error of trajectory 5 is the biggest. The reason is that 20 data points are not enough for FENN to grasp its periodicity

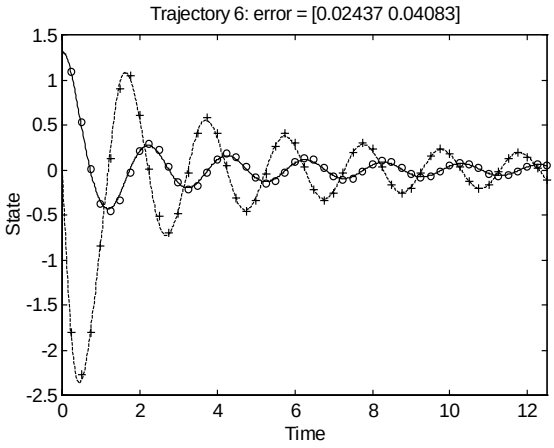

Figure 9

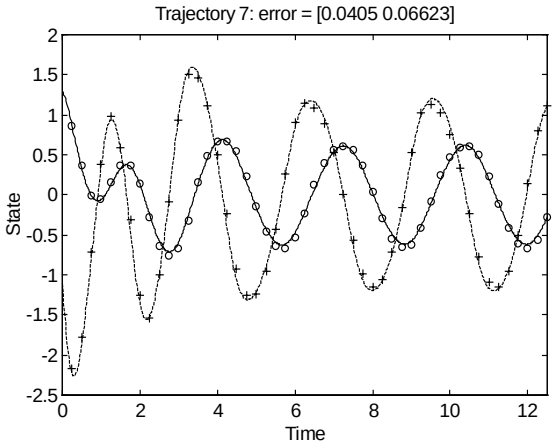

Figure 10

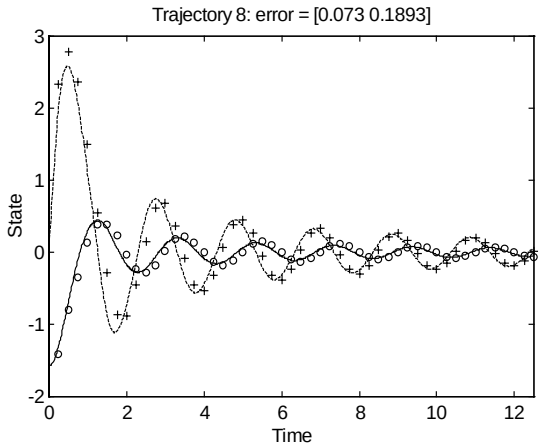

Figure 11

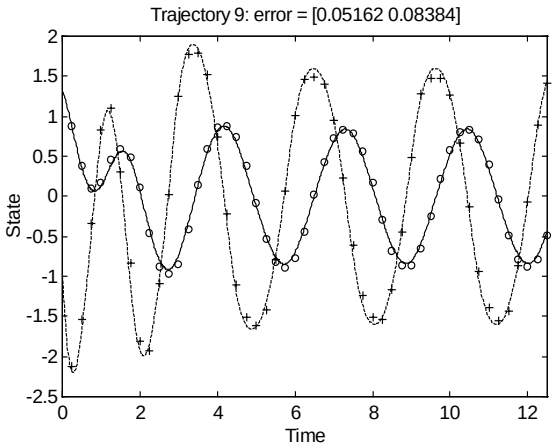

Figure 12

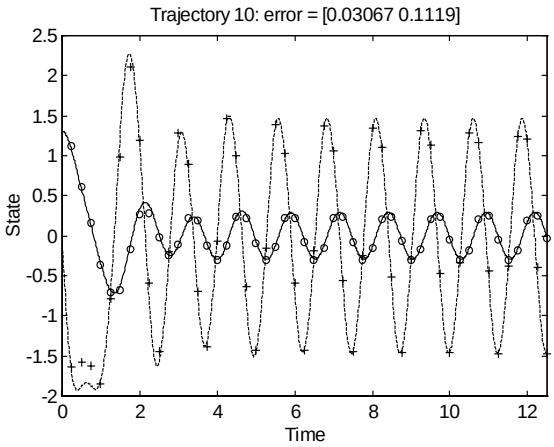

Figure 13

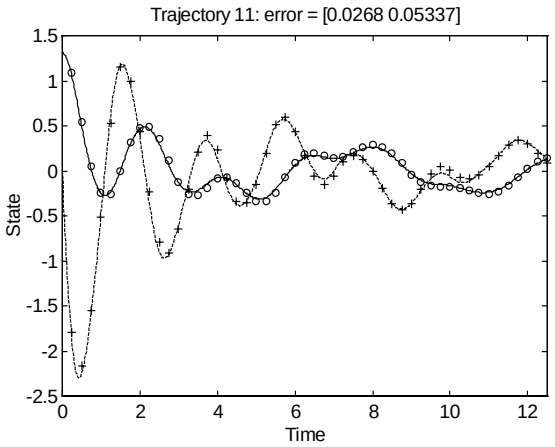

Figure 14

which is about 6*s*. The maximal error is taken place between time 5~6*s*, just after where FENN ends its learning.

- ✧ Trajectories 6, 7, 8 (Figure 9, Figure 10, Figure 11) give the example of starting the pendulum with some unlearned initial states. Though simple, the network does a good work.
- ✧ Trajectories 9 (Figure 12) and 11 (Figure 14) show that FENN can deal with the change in input amplitude as well as the initial state.

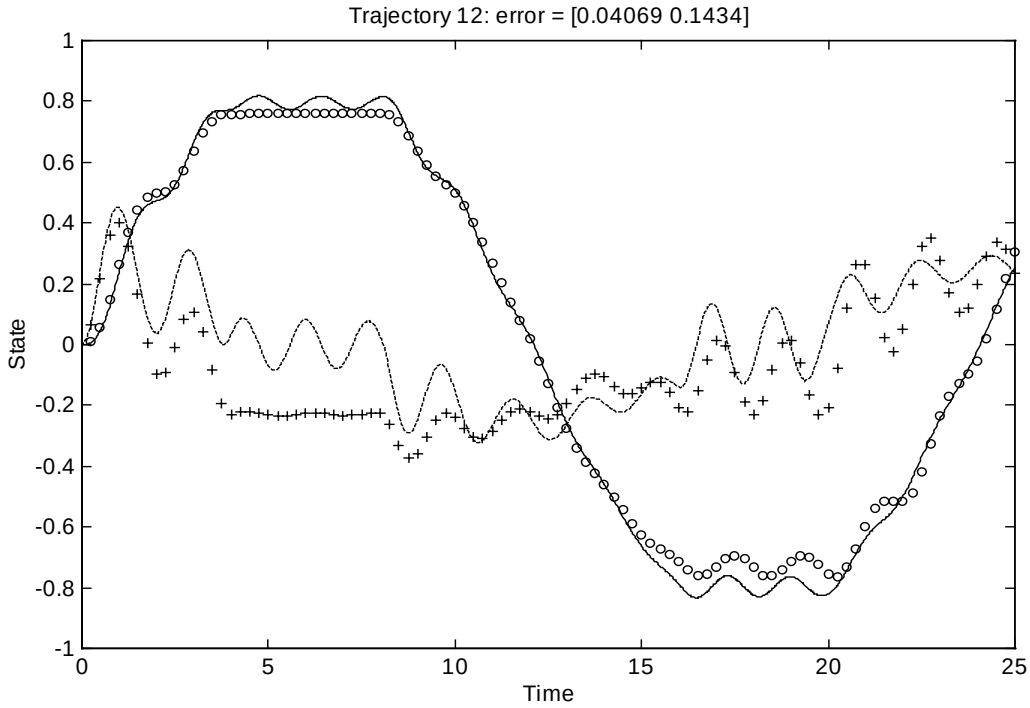

Figure 15 Simulation of trajectory 12 before membership learning

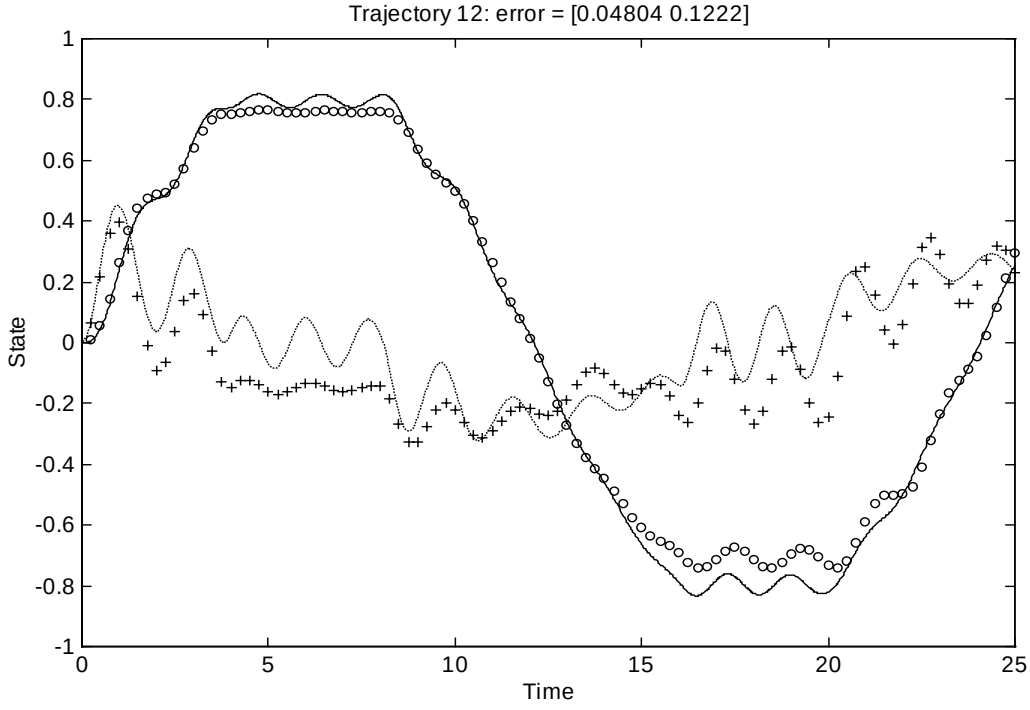

Figure 16 Trajectory 12 is better simulated by FENN after membership learning

- ✧ It is easy to notice that during the training we use force of sine type only with frequency 2 (rad/s), so the well simulation of trajectory 10 (Figure 13) which is under the force of $\sin(5t)$, is amazing. It should be pointed out that the generalization of frequency (from 2 to 5) is not intrinsically hold by the Elman-style networks [8]. This test shows FENN exhibits better generalization capability than the common Elman networks.
- ✧ Trajectory 12 (Figure 15) is a really difficult test to FENN. The force exerted to

the pendulum is at a very lower frequency than the characteristic one of the system, and the oscillating of pendulum seems with less order than previous trajectories. Though FENN does not give a perfect simulation on such condition, it really give the tendency of the pendulum, which is not easy.

After the rules have been extracted from the training data, we move to the second stage of learning – tuning the membership functions. After making the membership function parameters adjustable, FENN continues its learning for 606 cycles, and we get a little better test result (Table 2). We give another simulation of trajectory 12 in Figure 16. Compared to Figure 15, though still not very good, Figure 16 gives the two states (angle and speed) curves more closely to the desired ones, and the speed curve in latter simulation has clearly more similarities in the curve tendency than the former one. Reader would remember that trajectory 12 is not included in FENN's learning data, so this improving just reflects the improved generalization of FENN after the membership learning.

Table 3 gives the membership parameters (center and width) before and after such learning. Table 4 and Table 5 give the matrices A^r and B^r of all the nine fuzzy rules after the whole 1806-cycle learning, arranged in a big 3×3 matrix.

Table 2 Simulation errors

No.	Errors after 1200 cycles	Errors after another 606 cycles	No.	Errors after 1200 cycles	Errors after another 606 cycles
1	0.02361, 0.03170	0.02252, 0.05346	7	0.04050, 0.06623	0.03267, 0.05567
2	0.04170, 0.06655	0.03357, 0.05844	8	0.07300, 0.1893	0.08501, 0.2160
3	0.04631, 0.06702	0.03659, 0.05962	9	0.05162, 0.08384	0.04246, 0.06607
4	0.07048, 0.06216	0.06111, 0.05579	10	0.03067, 0.1119	0.03160, 0.1071
5	0.04115, 0.1242	0.04466, 0.1021	11	0.02680, 0.05337	0.02303, 0.04897
6	0.02437, 0.04083	0.02314, 0.04082	12	0.04069, 0.1434	0.04804, 0.1222

Table 3 Linguistic terms

state variable	x_1			x_2		
linguistic term	Negative	Zero	Positive	Negative	Zero	Positive
initial value (mid, wid)	-1.0472 0.5236	0.0000 0.5236	1.0472 0.5236	-2 1	0 1	2 1
learned value (mid, wid)	-1.0509 0.5256	0.0238 0.5582	1.0548 0.5227	-2.0197 0.9772	-0.0020 1.0337	2.0115 0.9840

5. CONCLUSION

The proposed FENN architecture exhibits several features:

- *The new form of fuzzy rules*. FENN employs the new form of fuzzy rules with linear state-space equations as the consequences. The linear state-space equations are convenient to human to represent dynamic systems, and in common sense such representation can grasp the inherent essential of the system. Natural and simple as the fuzzy set and fuzzy inference mechanism are to define the different aspects of a complex system, the nonlinearity of membership functions enables the network to simulate a nonlinear system well.

Table 4 Matrix $\boldsymbol{A}^r$ of all the nine rules after learning.

x_1 (θ) \ x_2 ($\dot{\theta}$)	Negative	Zero	Positive
Negative	$\begin{pmatrix} 0.2874 & 0.3629 \\ -0.2666 & -0.1577 \end{pmatrix}$	$\begin{pmatrix} 1.0646 & 0.2686 \\ -1.7172 & 0.6214 \end{pmatrix}$	$\begin{pmatrix} 0.4592 & -0.0860 \\ -0.7553 & 0.7015 \end{pmatrix}$
Zero	$\begin{pmatrix} 0.2336 & 0.1826 \\ -0.6706 & 0.4394 \end{pmatrix}$	$\begin{pmatrix} 0.8364 & 0.2932 \\ -2.4458 & 0.6865 \end{pmatrix}$	$\begin{pmatrix} 0.1761 & 0.1522 \\ -0.6387 & 0.4415 \end{pmatrix}$
Positive	$\begin{pmatrix} 0.3885 & -0.0665 \\ -0.7366 & 0.7671 \end{pmatrix}$	$\begin{pmatrix} 0.9584 & 0.0820 \\ -1.6243 & 0.7648 \end{pmatrix}$	$\begin{pmatrix} 0.3141 & 0.5258 \\ -0.2483 & -0.1755 \end{pmatrix}$

Table 5 Matrix $\boldsymbol{B}^r$ of all the nine rules after learning.

x_1 (θ) \ x_2 ($\dot{\theta}$)	Negative	Zero	Positive
Negative	$\begin{pmatrix} 0.0462 \\ 0.1156 \end{pmatrix}$	$\begin{pmatrix} -0.0287 \\ 0.0517 \end{pmatrix}$	$\begin{pmatrix} 0.0858 \\ 0.0481 \end{pmatrix}$
Zero	$\begin{pmatrix} 0.0311 \\ 0.1749 \end{pmatrix}$	$\begin{pmatrix} 0.0437 \\ 0.2675 \end{pmatrix}$	$\begin{pmatrix} 0.0312 \\ 0.1937 \end{pmatrix}$
Positive	$\begin{pmatrix} 0.0095 \\ 0.1629 \end{pmatrix}$	$\begin{pmatrix} 0.0039 \\ 0.0358 \end{pmatrix}$	$\begin{pmatrix} 0.0268 \\ 0.0737 \end{pmatrix}$

- *The context nodes*. The current state variables of the system are duplicated by the context nodes, and are used as part of the inputs at the next time. Like the Elman networks where the context layer copies the internal representation of the temporary inputs, the state vector happens to be the perfect form of internal representation of a dynamic system. This coincidence shows that the idea in the Elman networks and FENN are suitable for simulation of the dynamic systems.
- *Strong generalization capability*. In the simulation of the pendulum system, FENN shows a very strong generalization on different initial states and force inputs. This point can be deduced intuitively if we believe that FENN has the ability to grasp the inner qualities of the modeled system.
- *Rules presetting and easy extraction*. The *a priori* human knowledge can easily integrated into the network by the forms of fuzzy rules, and the learned rules can also easily extracted from the network. This is already discussed.

Though FENN shows a promising capability, there is still much work to do:

- *Automatic knowledge extraction*. In the simulation of the pendulum system, for its simplicity, we preset the crude membership functions and fix matrix $\boldsymbol{C}$ to unity. Though we can start learning from some random settings, it may be slow for FENN to converge and the learning may stick in local minima. So techniques like self-organization should be developed to automatically extract (crude) rules from the training data, before the time-consuming learning.
- *Stability analysis and assurance*. Since dynamic systems are involved, the stability of the network should be studied. There exist occasions that the network jump to infinity output during the learning. So both the stability principles of the network and the techniques to ensure the stability of the

learning should be explored.

- *Continuous form of FENN*. Though we can use a discrete one and interpolate among the discrete outputs, the continuous form of FENN may be more useful in real application, and is easy to fulfill with fuzzy chip or analog circuit. The network structure need no much change, but the learning algorithm should be rewritten to meet the continuous case.

References

[1] K.J. Hunt, D. Sbarbaro, R. Zbikowski and P.J. Gawthrop, "Neural networks for control systems — a survey," *Automatica* vol. **28** no. **6** (1992) 1083-1112.

[2] J.J. Buckley and Y. Hayashi, "Fuzzy neural networks: a survey," *Fuzzy Sets and Systems* **66** (1994) 1-13.

[3] J.J. Buckley and Y. Hayashi; "Fuzzy input-output controllers are universal approximators," *Fuzzy Sets and Systems* **58** (1993) 273-278.

[4] J.J. Buckley, "Sugeno type controllers are universal controllers," *Fuzzy Sets and Systems* **53** (1993) 299-304.

[5] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks* **4** (1991) 251-257.

[6] J.L. Elman, "Finding structure in time," *Cognitive Sci.* **14** (1990) 179-211.

[7] S.C. Kremer, "On the computational power of Elman-style recurrent networks," *IEEE Trans. Neural Networks* vol. **6** no. **4** (July 1995) 1000-1004.

[8] D.L. Wang, X.M. Liu and S.C. Ahalt, "On temporal generalization of simple recurrent networks," *Neural Networks* **9** (1996) 1099-1118.

[9] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modelling and control," *IEEE Trans. Systems Man Cybernet.* **15** (1985) 116-132.

[10] K. Yamashita, A. Suyitno and Y. Dote, "Stability analysis for classes of nonlinear fuzzy control systems," *Proc. of the IECON'93* vol. **1** (1993) 242-247.

[11] R.J. Williams and D. Zipser, "Experimental analysis of the real-time recurrent learning algorithm," *Connection Science* **1** (1989) 87-111.