Chaos Control using Universal Learning Network

Hirasawa, Kotaro
Department of Electrical and Electronic Systems Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University

Wang, Xiaofeng
Department of Electrical and Electronic Systems Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University : Graduate Student

Hu, Jinglu
Department of Electrical and Electronic Systems Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University

Murata, Junichi
Department of Electrical and Electronic Systems Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University

https://doi.org/10.15017/1498416

出版情報：九州大学大学院システム情報科学紀要. 4 (1), pp.13-27, 1999-03-26. 九州大学大学院システム情報科学研究科
バージョン：
権利関係：

九州大学大学院
システム情報科学研究科報告
第4巻 第1号 平成11年1月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.4, No.1, January 1999

Chaos Control using Universal Learning Network

Kotaro HIRASAWA*, Xiaofeng WANG**, Jinglu HU* and Junichi MURATA*

(Received December 21, 1998)

Abstract: Universal Learning Network(ULN) is proposed and its application to chaos control are discussed. ULNs form a super-set of neural networks. They consist of a number of inter-connected nodes where the nodes may have any continuously differentiable nonlinear functions in them and each pair of nodes can be connected by multiple branches with arbitrary (positive, zero, or even negative) time delays. A generalized learning algorithm is derived for the ULNs, in which both the first ordered derivatives (gradients) and the higher ordered derivatives are incorporated. The derivatives are calculated by using forward or backward propagation scheme. The algorithm can also be used in a unified manner for almost all kinds of learning networks. As an application of ULNs, a chaos control method using maximum Lyapunov number of ULNs is proposed. The maximum Lyapunov number of ULNs can be formulated by using higher ordered derivatives of ULNs and parameters of ULNs can be adjusted for the maximum Lyapunov number to approach the target value. From simulation results, it has been shown that a fully connected ULN with three nodes is able to display chaotic behaviors.

Keywords: Universal Learning Network,High ordered derivatives calculation,Chaos,Lyapunov number

1. Introduction

Since the first proposal of a neuron model by McCulloch and Pitts in 1940's, especially after the revitalization[1] of artificial neural networks in 1980's, a variety of neural networks have been devised and are now applied in many fields. The vast majority of neural networks in use are those networks whose parameters or weights are tuned by gradient-based supervised learning. This category includes feedforward networks or multilayer perceptrons[2], various types of recurrent neural networks[3),4)], radial basis function networks[5], fuzzy neural networks[6], and some networks with special architectures, such as time delay neural networks[7]. These networks seemingly have different architectures and are trained by distinguishable training algorithms. In essence, however, they can be unified in a single framework in regard to both their architectures and learning algorithms.

Universal Learning Networks (ULNs) is proposed, as the name indicates, to provide a universal framework for the class of neural networks. Unification of a variety of neural network architectures and their learning algorithms is an objective of ULNs; this provides a consistent viewpoint for the various kinds of neural networks. However, there is another benefit that is expected for ULNs. Generalization of the architectures and the gradient-based algorithms does not only give a unified description but attains new abilities of the networks. Allowing high degrees of freedom in their architectures gives them more flexibilities and representing abilities, and therefore the ULNs can be useful and effective tools for modeling and controlling large-scale nonlinear complex systems including physical, social and economical phenomena. In addition to calculation of the first ordered derivatives of the signals flowing in the networks that are necessary in gradient-based learning, the generalized ULN learning algorithm is equipped with a systematic mechanism that can calculates their second or higher ordered derivatives. This allows elaborate criterion functions to be used in their learning, which enables more sophisticated specification of the network performance[8].

Before going into the details, it should be worth giving to the readers, as preliminary knowledge, the salient features of the ULNs and their differences from the existing alternatives.

A ULN consists of a number of inter-connected nodes. They may have any continously differentiable functions in them, and each pair of nodes can be connected by multiple branches with arbitrary (positive, zero, or even negative) time delays. Networks having time delays of multiple-length can be converted to those having unit-length time delay

* Department of Electrical and Electronic Systems Engineering

** Department of Electrical and Electronic Systems Engineering, Graduate Student

only by inserting an appropriate number of additional nodes between the nodes. However, zero or negative time delays can not be represented by the unit time delays. A ULN including negative time delays between the nodes can be used to model the systems with prediction mechanism.

Learning of a ULN is defined as determination of its optimal parameter values that minimize a criterion function. Generalized learning algorithms are derived based on two ideas. One is that the static networks are merely special dynamic networks, and the other is that like the method proposed by Williams[3),4)] the derivatives of the node outputs can be obtained by either propagating the changes of the node outputs caused by the changes of the parameters through the network forward in time and space or propagating the change of the criterion function caused by the changes of the node outputs backward in time and space.

The most important feature of the learning of ULN is use of the higher ordered derivatives of the criterion function with respect to parameters. Buntine[8)] summarized the methods for calculating the second ordered derivatives in feedforward neural networks but in ULNs a generalized calculation method of the n-th ordered derivatives even for the recurrent networks is proposed.

The ULNs allows any nonlinear functions to be embeded in their nodes. Therefore, both a controlled system and its controller can be represented by a single ULN. If the controlled system is unknown, its identification can also be done by the ULN learning algorithm[9)]. Then an optimal controller can be obtained by off-line gradient learning.

The ULN controllers have a similarity with traditional nonlinear optimal controllers in the sense that both are constructed by minimizing criterion functions or performance indexes. However, the control laws obtained by the traditional controller design methods are, in most cases, feedforward control laws, and the control signals are given as functions of time. On the other hand, the ULN controllers form feedback control schemes and their parameters are determined by optimization. Therefore, although the ULN controllers do not attain better performance indexes because of their structural constraint, they exhibit better robustness against external disturbances due to their feedback configuration.

The higher ordered derivative calculation mechanism of the ULNs renders additional advantages to the ULN controllers. As a criterion function, one may employ a standard control performance index, for example, tracking error of the controlled system. In addition to this, one can put other terms to their criterion functions. Let us assume that our additional term is the sensitivity of the standard criterion function with respect to the changes in the plant parameters. Minimization of the sensitivity by a gradient method requires the second ordered derivatives of the standard criterion function since the sensitivity itself is the first ordered derivative. This derivative can be calculated by the proposed mechanism. Minimizing this extended criterion function will result in a robust controlled system that is not much affected by changes in the plant parameters.

Along with the development of neural networks, the chaotic behavior, one of the typical characteristics of biological neural networks has been dicussed in many papers. For example, Aihara's chaotic neural networks[10)], Chen's chaotic simulated annealing by a neural network model with transient chaos[11)], Falcioni's correlation functions and relaxation properties in chaotic dynamics and statistical mechanics[12)], Ikeda's high-dimensional chaotic behavior in systems with time-delay feedback[13)], etc. But all these papers use special nonlinear functions or special systems to model chaotic phenomena, and main characteristics of chaotic behavior, Lyapunov number, is only used to analyze the systems,not to control the systems. And other famous paper[17)] for controlling chaos is only dealing with the method to eliminate the chaotic phenomena, not to generate the chaos.

It is generally well known that chaotic behaviors are characterized mainly by Lyapunov numbers of the dynamic systems. If one or more Lyapunov numbers are greater than zero, in other words, the maximum Lyapunov number is positive, then the system behaves chaotically, otherwise, the system is stable. Therefore, it is highly motivated to develop a method to control the Lyapunov numbers in dynamic systems. In this paper,we propose a new method to control chaotic behavior of dynamic systems by using the maximum Lyapunov number and higher ordered derivatives of ULN.

The reason why the higher ordered derivatives of ULN is used is that the maximum Lyapunov number itself is expressed by the first ordered derivatives, therefore the second order derivatives are needed to differentiate the maximum Lyapunov number when adjusting the parameters by the gradinet method. And the proposed method has

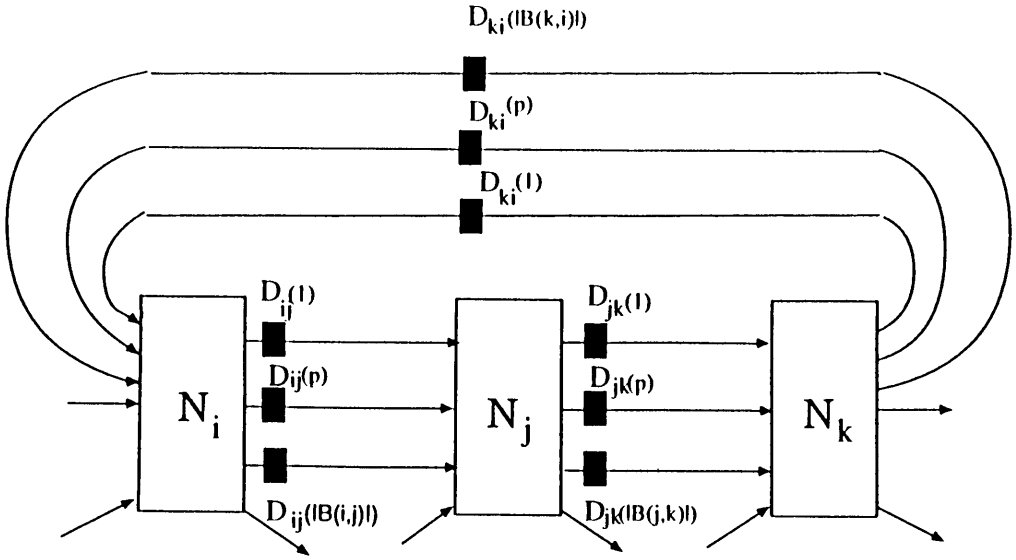

Fig.1 Structure of ULNs

a distinctive feature that any special mechanism is not needed to control the chaos because the maximum Lyapunov number is adopted as the criterion function.

This paper is organized as follows. In the next section, the structure and the learning algorithms of ULNs are described. The specific procedures for calculating the higher ordered derivatives are presented in sections 3 and 4. Section 3 proposes the backward propagation procedure, while setion 4 describes the forward propagation scheme and compares the computational complexities of the both procedures. In section 5, a new chaos control method is proposed, where maximum Lyapunov number of the dynamic systems approaches its target value by adjusting the parameters of the systems using the second ordered derivatives of the ULNs. Generation and die-out of the chaotic phenomena are shown in section 6 which describes the simulation results. Section 7 gives conclusions.

2. Universal Learning Networks

In this section, we discuss the structure and learning algorithm of ULNs.

2.1 Structure of ULNs

As shown in **Fig.1**, a ULN consists of a number of nodes and branches for inter-connecting the nodes. The nodes may have any continously differentiable nonlinear functions in them, and each pair of nodes can be connected each other by multiple branches with arbitrary (positive, zero or even negative) time delays. ULNs including negative time delays can be used to model systems that have prediction mechanism in them.

The generic equation that describes the behavior of ULNs is expressed as follows,

$$h_j(t) = f_j(\{h_i(t - D_{ij}(p)) | i \in JF(j), p \in B(i,j)\}, \{\lambda_m(t) \mid m \in M(j)\}, \{r_n(t) \mid n \in N(j)\}) \quad j \in J, t \in T, \tag{1}$$

where

$h_j(t)$	:	output value of node j at time t , $j \in J$
$\lambda_m(t)$	:	value of m-th parameter at time t, $m \in M(j)$
$r_n(t)$	:	value of n-th external input variable at time t $n \in N(j)$
f_j	:	nonlinear function of node j
$D_{ij}(p)$	:	time delay of p-th branch from node i to node j
J	:	set of nodes $\{j\}$
$JF(j)$	:	set of nodes which are connected to node j
N	:	set of external input variables $\{n\}$
$N(j)$	:	$\{n \mid r_n$ is fed to node $j\}$
M	:	set of parameters $\{m\}$
$M(j)$	:	$\{m \mid h_j$ is partially differentiable with respect to $\lambda_m\}$
$B(i,j)$	:	set of branches from node i to node j
T	:	discrete set of sampling times.

The ULNs operate on a diecrete-time basis. Each

pair of nodes i and j may have multiple branches between them, i.e, set $B(i,j)$ may contain more than one element, and parameters λ_m can be time-varying. Function $f_j(*)$ that governs the operation of nodes can be any contionously differentiable function.

2.2 Learning of ULNs

Learning of ULNs is realized by minimizing a criterion function L based on the gradient method.

$$\lambda_1 \leftarrow \lambda_1 - \gamma \frac{\partial^\dagger L}{\partial \lambda_1} \tag{2}$$

where γ is the learning coefficient assigned with a small positive value and $\frac{\partial^\dagger L}{\partial \lambda_1}$ is the orderd derivative defined by Werbos in 14) which means the change of the criterion function L caused by the change of λ_1 with other variables being fixed.

The criterion function L usually consists of two parts, a fundamental part E and an extened part E_x,

$$L = E + E_x \tag{3}$$

The fundamental part E is defined as a function of any node outputs, any parameters and any time instants,

$$E = E(\{h_r(s)\}, \{\lambda_m\}) \quad r \in J_0, m \in M_0, s \in T_0 \tag{4}$$

where

J_0 : $\{r|h_r$ is related to evaluation of criterion function $\}$

M_0 : $\{m|\lambda_m$ is related to evaluation of criterion function$\}$

T_0 : $\{s|$ time instant s is related to evaluation of criterion function$\}$

A typical choice of E is sum of errors between the network outputs and their desired values.

The extended part E_x is a function of derivatives of node output $h_r(s)$. Therefore, this part is related to notions that can not be represented by fundamental criterion E, such as smoothness, robustness and stability, thus it renders the ULNs a variety of advantageous features. The cost that one might have to pay for these advantages is computational complexity in the gradient-based optimization of the generalized criterion function L. The gradient of L includes not only the first ordered derivatives of node output $h_r(s)$ but also its higher ordered derivatives since E_x in L itself contains the derivatives of $h_r(s)$. In the subsequent sections, we will propose systematic calculation methods of the derivatives. Of course, we have to endure the additional computational cost for calculation of the higher ordered derivatives. However, the calculation itself is not complicated and is easily implemented. Since E is a differertiable function of node output $h_r(s)$, calculation of the derivatives of E and the computation of those of $h_r(s)$ are essentially identical. Therefore, for clarity, in the following two sections we will discuss the first and higher ordered derivatives of E only. Their computational complexity will be evaluated and compared in section 4.

3. Derivatives Calculation Method by Backward Propagation

In this section, a backward calculation method of the first ordered derivatives and the second ordered derivatives of the criterion function E (Eq.(4)) with respect to the parameters will be presented, which can be easily extended to calculation of the n-th ordered derivatives.

3.1 The First Ordered Derivatives

When the idea of backpropagation was first presented in 1972, they expressed legitimate concern about the validity of the ranther complex calculations involved. To deal with this problem, Werbos proved a new chain rule for ordered derivatives:

$$\frac{\partial^\dagger TARGET}{\partial z_i} = \sum_{j>i} \frac{\partial^\dagger TARGET}{\partial z_j} \times \frac{\partial z_j}{\partial z_i} + \frac{\partial TARGET}{\partial z_i} \tag{5}$$

where the derivatives with the superscript represent ordered derivatives, and the derivatives without subscripts represent ordinary partial derivatives. This chain rule is valid only for ordered system where the values to be calculated can be calculated one by one (if necessary) in the order $z_1, z_2, \ldots, z_n$, TARGET. The simple partial derivatives represent the direct impact of z_i on z_j through the system equation which determines z_j. The ordered derivatives represent the total impact of z_i on TARGET, accounting for both the direct and indirect effects. Let us evaluate the change of criterion function E caused by a time-varying parameter $\lambda_1(t)$ at time $t = t_1$. When TARGET is crition function E, z_i is parameter $\lambda_1(t_1)$, z_j is intermediate variable $h_d(t_1)$, the first ordered derivative of

criterion function E with respcet to $\lambda_1(t_1)$ can be calculated through Eq.(6), (7), (8).

$$\frac{\partial^\dagger E}{\partial \lambda_1(t_1)} = \sum_{d \in JD(\lambda_1)} [\frac{\partial^\dagger E}{\partial h_d(t_1)} \frac{\partial h_d(t_1)}{\partial \lambda_1(t_1)}] + \frac{\partial E}{\partial \lambda_1(t_1)} \quad (6)$$

where

t_1	:	a certain specified time instant
$\lambda_1(t_1)$	:	value of parameter at time t_1
$JD(\lambda_1)$	:	$\{d \mid h_d(t_1)$ is partially differen -tiable with respect to $\lambda_1(t_1)\}$

Since node output $h_d(t_1)$ may affect E directly or indirectly, we need to consider the ordered derivative $\frac{\partial^\dagger E}{\partial h_d(t_1)}$ here. Now, let us denote the ordered derivative as $\delta_1(j,t)$

$$\delta_1(j,t) = \frac{\partial^\dagger E}{\partial h_j(t)} \quad (7)$$

Its evaluation again requires the help of some intermediate variables, which leads to the following backward propagation algorithm,

$$\delta_1(j,t) = \sum_{k \in JB(j)} \sum_{p \in B(j,k)} [\frac{\partial h_k(t + D_{jk}(p))}{\partial h_j(t)} \delta_1(k, t + D_{jk}(p))] + \frac{\partial E}{\partial h_j(t)} \quad (8)$$
$$j \in J, t \in T$$

The first term in the right-hand side indicates the indirect effect which is calculated taking outputs of the *downstream* nodes k as the intermediate variables, and the second term shows the direct effect of node output $h_j(t)$ on the criterion E.

Substituting this signal $\delta_1(j,t)$ into Eq.(6), we can derive the first ordered derivative of E with respective to time-varying parameter λ_1 at time t_1, and also the derivative with respective to a time-invariant parameter λ_1 can be obtained in a similar way. For static networks, the formulas are reduced to simple forms. The calculation procedures of the first ordered derivatives for time-varying parameters, time-invariant parameters and static networks are summarized as follows,

- For time-varying parameters

$$\frac{\partial^\dagger E}{\partial \lambda_1(t_1)} = \sum_{d \in JD(\lambda_1)} [\frac{\partial h_d(t_1)}{\partial \lambda_1(t_1)} \delta_1(d, t_1)] + \frac{\partial E}{\partial \lambda_1(t_1)} \quad (9)$$

- For time-invariant parameters

$$\frac{\partial^\dagger E}{\partial \lambda_1} = \sum_{t' \in T} \sum_{d \in JD(\lambda_1)} [\frac{\partial h_d(t')}{\partial \lambda_1} \delta_1(d, t')] + \frac{\partial E}{\partial \lambda_1} \quad (10)$$

- For static networks

$$\frac{\partial^\dagger E}{\partial \lambda_1} = \sum_{d \in JD(\lambda_1)} [\frac{\partial h_d}{\partial \lambda_1}] + \frac{\partial E}{\partial \lambda_1} \quad (11)$$

$$\delta_1(j) = \sum_{k \in JB(j)} [\frac{\partial h_k}{\partial h_j} \delta_1(k)] + \frac{\partial E}{\partial h_j} \quad (12)$$
$$j \in J$$

The summation over $d \in JD(\lambda_1)$ in Eq.(9),(10) and (11) reflects the fact that there a number of nodes whose outputs are partially differentiable with respect to a certain λ_1. Although a change of time-varying parameter at a specific time instant t_1 directly affects node outputs $h_d(t)$ at time $t = t_1$ only, a change in a time-invariant parameter directly cause change of $h_d(t)$ at any time. Therefore, we need to consider $\frac{\partial h_d(t')}{\partial \lambda_1}$ at any time t' and sum them up. This is the reason why Eq.(10) only contains a summation over t'. In the algorithm (8)-(12), the derivative $\frac{\partial^\dagger E}{\partial \lambda_1}$ is calculated by propagating δ_1 or the derivative of the function E with respect to the node outputs backward in time (for dynamic networks) or backward in space (for static networks), which is, in essence, identical to the back-propagation algorithm for ordinary neural networks. In other words, the learning algorithm includes the ordinary back-propagation as one of its special examples.

3.2 The Second Ordered Derivatives

The first ordered derivative calculation discussed in the preceding subsection can be easily extended to calculation of the n-th ordered derivatives. Details of the n-th ordered derivative calculation method by backward propagation can be found in [15]. In this subsection, we give the calculation method of the second ordered derivatives as an example.

Let t_1 and t_2 be certain specified time instants, λ_1 and λ_2 be parameters. Then by differentiating Eq.(9) with respect to $\lambda_2(t_2)$, we have the second

ordered derivative $\frac{\partial^{\dagger 2}E}{\partial\lambda_1(t_1)\lambda_2(t_2)}$ as,

$$\frac{\partial^{\dagger 2}E}{\partial\lambda_1(t_1)\lambda_2(t_2)} = \sum_{d\in JD(\lambda_1)}[\frac{\partial h_d(t_1)}{\partial\lambda_1(t_1)}\frac{\partial^{\dagger}\delta_1(d,t_1)}{\partial\lambda_2(t_2)} + \frac{\partial^{\dagger}(\frac{\partial h_d(t_1)}{\partial\lambda_1(t_1)})}{\partial\lambda_2(t_2)}\delta_1(d,t_1)] + \frac{\partial^{\dagger}(\frac{\partial E}{\partial\lambda_1(t_1)})}{\partial\lambda_2(t_2)} \quad (13)$$

Let us define $\delta_{12}(d,t_1) = \frac{\partial^{\dagger}\delta_1(d,t_1)}{\partial\lambda_2(t_2)} = \frac{\partial^{\dagger}(\partial^{\dagger}E/\partial_{h_d}(t_1))}{\partial\lambda_2(t_2)}$. It can be calculated by differentiating (8) with respect to $\lambda_2(t_2)$

$$\lambda_{12}(j,t) = \sum_{k\in JB(j)}\sum_{p\in B(j,k)}[\frac{\partial h_k(t+D_{jk}(p))}{\partial h_j(t)} \times\delta_{12}(t+D_{jk}(p))] + \sum_{k\in JB(j)}\sum_{p\in B(j,k)}[\frac{\partial^{\dagger}(\frac{\partial h_k(t+D_{jk}(p))}{\partial h_j(t)})}{\partial\lambda_2(t_2)} \times\delta_1(k,t+D_{jk}(p))] + \frac{\partial^{\dagger}(\frac{\partial E}{\partial h_j(t)})}{\partial\lambda_2(t_2)} \quad (14)$$
$$j\in J, t\in T$$

In Eq.(13) and (14), $\frac{\partial^{\dagger}(\partial h_d(t_1)/\partial\lambda_1(t_1))}{\partial\lambda_2(t_2)}$, $\frac{\partial^{\dagger}(\partial E/\partial\lambda_1(t_1))}{\partial\lambda_2(t_2)}$, $\frac{\partial^{\dagger}(\partial h_k(t+D_{jk}(p))/\partial h_j(t))}{\partial\lambda_2(t_2)}$ and $\frac{\partial^{\dagger}(\partial E/\partial h_j(t))}{\partial\lambda_2(t_2)}$ can be calculated by applying the calculation procedure of the first ordered derivatives, with E being replaced with $\frac{\partial h_d(t_1)}{\partial\lambda_1(t_1)}$, $\frac{\partial E}{\partial\lambda_1(t_1)}$, $\frac{\partial h_k(t+D_{jk}(p))}{\partial h_j(t)}$ and $\frac{\partial E}{\partial h_j(t)}$, respectively. For time invariant parameters and static ULNs, the calculation procedure is rewritten as follows,

- For time-invariant parameters

$$\frac{\partial^{\dagger 2}E}{\partial\lambda_1\partial\lambda_2} = \sum_{t'\in T}\sum_{d\in JD(\lambda_1)}[\frac{\partial h_d(t')}{\partial\lambda_1}\delta_{12}(d,t') + \frac{\partial^{\dagger}(\frac{\partial h_d(t')}{\partial\lambda_1})}{\partial\lambda_2}\delta_1(d,t')] + \frac{\partial^{\dagger}(\frac{\partial E}{\partial\lambda_1})}{\partial\lambda_2} \quad (15)$$

$$\delta_{12}(j,t) = \sum_{k\in JB(j)}\sum_{p\in B(j,k)}[\frac{\partial h_k(t+D_{jk}(p))}{\partial h_j(t)} \times\delta_{12}(k,t+D_{jk}(p))] + \sum_{k\in JB(j)}\sum_{p\in B(j,k)}[\frac{\partial^{\dagger}(\frac{\partial h_k(t+D_{jk}(p))}{\partial h_j(t)})}{\partial\lambda_2} \times\delta_1(k,t+D_{jk}(p))] + \frac{\partial^{\dagger}(\frac{\partial E}{\partial h_j(t)})}{\partial\lambda_2} \quad (16)$$
$$j\in J, t\in T$$

- For static networks

$$\frac{\partial^{\dagger 2}E}{\partial\lambda_1\partial\lambda_2} = \sum_{d\in JD(\lambda_1)}[\frac{\partial h_d}{\partial\lambda_1}\delta_{12}(d) + \frac{\partial^{\dagger}(\frac{\partial h_d}{\partial\lambda_1})}{\partial\lambda_2}\delta_1(d)] + \frac{\partial^{\dagger}(\frac{\partial E}{\partial\lambda_1})}{\partial\lambda_2} \quad (17)$$

$$\delta_{12}(j) = \sum_{k\in JB(j)}[\frac{\partial h_k}{\partial h_j}\delta_{12}(k) + \frac{\partial^{\dagger}(\frac{\partial h_k}{\partial h_j})}{\partial\lambda_2}\delta_1(k)] + \frac{\partial^{\dagger}(\frac{\partial E}{\partial h_j})}{\partial\lambda_2} \quad (18)$$
$$j\in J$$

The n-th ordered derivative $\frac{\partial^{\dagger n}E}{\partial\lambda_1(t_1)\partial\lambda_2(t_2)\ldots\partial\lambda_n(t_n)}$ can be obtained by evaluating iterative equations for $\delta_1, \delta_{12}, \delta_{13}\ldots, \delta_{12\ldots n} = \frac{\partial^{\dagger}\delta_{12\ldots n-1}}{\partial\lambda_n}$. Details of the calculation method of n-th ordered derivatives can be found in [15]. Eq.(17) and Eq.(18) are generalized versions of Buntines method in [8].

4. Derivatives Calculation Method by Fordward Propagation

In this section , we will introduce a new calculation scheme for the n-th ordered derivatives of the criterion function E by using forward propagation method. In the forward propagation calculation, the change of node outputs with respect to changes in the parameters propagate forwards, whereas the changes of criteron function with respective to changes in node outputs propagate backwards in the backward propagation calculation. Details of the n-th ordered derivatives calculation by forward propagation method can be found in [16]. In this section, only the calculation of the first and the second ordered derivatives are discussed as examples.

4.1 The First Ordered Derivative

In contrast to the backward propagation procedure described in the previous section, let us think of evaluating the indirect effect of the changes in the parameter of the criterion function E by considering the node outputs that directly influence E as the intermediate variables. Then, the first ordered derivative of the function E with respect to a time-varying parameter $\lambda_1(t_1)$ at time t_1 can be calculated as follows,

$$\frac{\partial^{\dagger}E}{\partial\lambda_1(t_1)} = \sum_{r\in J_0}\sum_{s\in T_0}[\frac{\partial E}{\partial h_r(s)}\frac{\partial^{\dagger}h_r(s)}{\partial\lambda_1(t_1)}] + \frac{\partial E}{\partial\lambda_1(t_1)} \quad (19)$$

where

t_1 : a certain specified time instant
$\lambda_1(t_1)$: value of parameter at time t_1
J_0 : $\{\ r|h_r$ is related to evaluation of $E\ \}$
T_0 : $\{\ s|$ time instant s at which E is evaluated $\}$

Eq.(19) shows that the change of criterion function caused by the change of parameters can be reduced to the change of the outputs of nodes related to the evaluation of criterion function. The ordered derivative of node output $h_j(t)$ with respect to the time-varying parameter $\lambda_1(t_1)$,

$$P_1(j,t,\lambda_1(t_1)) = \frac{\partial^\dagger h_j(t)}{\partial \lambda_1(t_1)} \tag{20}$$

can be calculated using the following forward propagation procedure

$$\begin{aligned} P_1(j,t,\lambda_1(t_1)) = & \sum_{i\in JF(j)} \sum_{p\in B(i,j)} [\frac{\partial h_j(t)}{\partial h_i(t-D_{ij}(p))} \\ & \times P_1(i,t-D_{ij}(p),\lambda_1(t_1))] \\ & + \frac{\partial h_j(t)}{\partial \lambda_1(t_1)} \\ & j\in J, t\in T \end{aligned} \tag{21}$$

The first term in the right-hand side indicates the indirect effect which is calculated taking the outputs of the *upstream* nodes i as intermediate variables, while the second term shows the direct effect of parameter value $\lambda_1(t_1)$ on node output $h_j(t)$. For time-invariant parameters and static networks, Eq.(19) and (21) can be reduced to the following simple forms.

- For time-invariant parameters

$$\frac{\partial^\dagger E}{\partial \lambda_1} = \sum_{r\in J_0} \sum_{s\in T_0} [\frac{\partial E}{\partial h_r(s)} P_1(r,s,\lambda_1)] + \frac{\partial E}{\partial \lambda_1} \tag{22}$$

$$\begin{aligned} P_1(j,t,\lambda_1) = & \sum_{i\in JF(j)} \sum_{p\in B(i,j)} [\frac{\partial h_j(t)}{\partial h_i(t-D_{ij}(p))} \\ & \times P_1(i,t-D_{ij}(p),\lambda_1)] \\ & + \frac{\partial h_j(t)}{\partial \lambda_1} \\ & j\in J, t\in T \end{aligned} \tag{23}$$

where $P_1(j,t,\lambda_1) = \frac{\partial^\dagger h_j(t)}{\partial \lambda_1}$.

- For static networks

$$\frac{\partial^\dagger E}{\partial \lambda_1} = \sum_{r\in J_0} [\frac{\partial E}{\partial h_r} P_1(r,\lambda_1)] + \frac{\partial E}{\partial \lambda_1} \tag{24}$$

$$P_1(j,\lambda_1) = \sum_{i\in JF(j)} [\frac{\partial h_j}{\partial h_i} P_1(i,\lambda_1)] + \frac{\partial h_j}{\partial \lambda_1} \tag{25}$$

$$j\in J$$

where $P_1(j,\lambda_1) = \frac{\partial^\dagger h_j}{\partial \lambda_1}$.

In the backward propagation scheme, the calculation of the first ordered derivative of E with respective to a time-invariant parameter λ_1 needed a summation over $t'\in T$ (see Eq.(9)). This was because that a change in a time-invariant parameter directly causes change of $h_d(t)$ at any time. In contrast to this, the forward propagation procedure for the time-invariant parameters does not contain such a summation. The reason for this is that, in the forward propagation scheme, the sensitivity of a node output $h_r(s)$ with respect to the change in a parameter λ_1 is evaluated by an ordered derivative which incorporates all the direct and indirect effects. The indirect effects are calculated by following a chain of spatial and temporal causes-and-effects as shown in Eq.(21), (23) and (25). Therefore, $P_1(j,t,\lambda_1)$ includes all the effects at any time $t'\le t$ caused by the change of λ_1. Eq.(24) and Eq.(25) are generalized equations of William's Real Time Recurrent Learning (RTRL)[4].

4.2 The Second Ordered Derivatives

Let t_1 and t_2 be certain specified time instants, and $\lambda_1(t)$ and $\lambda_2(t)$ be time-varying parameters. Then the second ordered derivative $\frac{\partial^{\dagger 2} E}{\partial \lambda_1(t_1)\partial \lambda_2(t_2)}$ can be obtained by differentiating Eq.(19) with respect to $\lambda_2(t_2)$.

$$\begin{aligned} \frac{\partial^{\dagger 2} E}{\partial \lambda_1(t_1)\lambda_2(t_2)} = & \sum_{r\in J_0} \sum_{s\in T_0} [\frac{\partial^\dagger (\frac{\partial E}{\partial h_r(s)})}{\partial \lambda_2(t_2)} \frac{\partial^\dagger h_r(s)}{\partial \lambda_1(t_1)} \\ & + \frac{\partial E}{\partial h_r(s)} \frac{\partial^{\dagger 2} h_r(s)}{\partial \lambda_1(t_1)\partial \lambda_2(t_2)}] \\ & + \frac{\partial^\dagger (\frac{\partial E}{\partial \lambda_1(t_1)})}{\partial \lambda_2(t_2)} \end{aligned} \tag{26}$$

Then introducing

$$\begin{aligned} P_2(j,t,\lambda_1(t_1),\lambda_2(t_2)) &= \frac{\partial^{\dagger 2} h_j(t)}{\partial \lambda_1(t_1)\partial \lambda_2(t_2)} \\ &= \frac{\partial^\dagger P_1(j,t,\lambda_1(t_1))}{\partial \lambda_2(t_2)} \end{aligned} \tag{27}$$

and differentiating Eq.(21) with respect to $\lambda_2(t_2)$, the following iterative formula for P_2 can be obtained,

$$
\begin{aligned}
&P_2(j,t,\lambda_1(t_1),\lambda_2(t_2)) \\
&= \sum_{i\in JF(j)}\sum_{p\in B(i,j)}[\frac{\partial^\dagger(\frac{\partial h_j(t)}{\partial h_i(t-D_{ij}(p))})}{\partial\lambda_2(t_2)} \\
&\quad \times P_1(i,t-D_{ij}(p),\lambda_1(t_1))] \\
&+ \sum_{i\in JF(j)}\sum_{p\in B(i,j)}[\frac{\partial h_j(t)}{\partial h_i(t-D_{ij}(p))} \\
&\quad \times P_2(i,t-D_{ij}(p),\lambda_1(t_1),\lambda_2(t_2))] \\
&+ \frac{\partial^\dagger(\frac{\partial h_j(t)}{\partial\lambda_1(t_1)})}{\partial\lambda_2(t_2)} \qquad (28) \\
&\qquad j\in J, t\in T
\end{aligned}
$$

The first ordered derivatives in Eq.(26) and (28), for example $\frac{\partial^\dagger(\partial E/\partial h_r(s))}{\partial\lambda_2(t_2)}$, can be calculated by replacing $\frac{\partial E}{\partial h_r(s)}$ with E and using the first order derivative calculation again.

Furthermore, as in the preceding section, Eq.(26) and (28) can be transformed to the following simplified expressions for time-invariant parameters and static networks.

- For time-invariant parameters

$$
\begin{aligned}
\frac{\partial^{\dagger 2}E}{\partial\lambda_1\partial\lambda_2} &= \sum_{r\in J_0}\sum_{s\in T_0}[\frac{\partial^\dagger(\frac{\partial E}{\partial h_r(s)})}{\partial\lambda_2}P_1(r,s,\lambda_1) \\
&\quad + \frac{\partial E}{\partial h_r(s)}P_2(r,s,\lambda_1,\lambda_2)] \\
&\quad + \frac{\partial^\dagger(\frac{\partial E}{\partial\lambda_1})}{\partial\lambda_2} \qquad (29)
\end{aligned}
$$

$$
\begin{aligned}
P_2(j,t,\lambda_1,\lambda_2) &= \sum_{i\in JF(j)}\sum_{p\in B(i,j)}[\frac{\partial^\dagger(\frac{\partial h_j(t)}{\partial h_i(t-D_{ij}(p))})}{\partial\lambda_2} \\
&\quad \times P_1(i,t-D_{ij}(p),\lambda_1)] \\
&\quad + \sum_{i\in JF(j)}\sum_{p\in B(i,j)}[\frac{\partial h_j(t)}{\partial h_i(t-D_{ij}(p))} \\
&\quad \times P_2(i,t-D_{ij}(p),\lambda_1,\lambda_2)] \\
&\quad + \frac{\partial^\dagger(\frac{\partial h_j(t)}{\partial\lambda_1})}{\partial\lambda_2} \qquad (30) \\
&\qquad j\in J, t\in T
\end{aligned}
$$

where $P_2(j,t,\lambda_1,\lambda_2) = \frac{\partial^{\dagger 2}h_j(t)}{\partial\lambda_1\partial\lambda_2}$.

- For static networks

$$
\begin{aligned}
\frac{\partial^{\dagger 2}E}{\partial\lambda_1\partial\lambda_2} &= \sum_{r\in J_0}[\frac{\partial^\dagger(\frac{\partial E}{\partial h_r})}{\partial\lambda_2}P_1(r,\lambda_1) \\
&\quad + \frac{\partial E}{\partial h_r}P_2(r,\lambda_1,\lambda_2)] \\
&\quad + \frac{\partial^\dagger(\frac{\partial E}{\partial\lambda_1})}{\partial\lambda_2} \qquad (31)
\end{aligned}
$$

$$
\begin{aligned}
P_2(j,\lambda_1,\lambda_2) &= \sum_{i\in JF(j)}[\frac{\partial^\dagger(\frac{\partial h_j}{\partial h_i})}{\partial\lambda_2}P_1(i,\lambda_1) \\
&\quad + \frac{\partial h_j}{\partial h_i}P_2(i,\lambda_1,\lambda_2)] \\
&\quad + \frac{\partial^\dagger(\frac{\partial h_j}{\partial\lambda_1})}{\partial\lambda_2} \qquad (32) \\
&\qquad j\in J
\end{aligned}
$$

where $P_2(j,\lambda_1,\lambda_2) = \frac{\partial^{\dagger 2}h_j}{\partial\lambda_1\partial\lambda_2}$.

The n-th order derivative $\frac{\partial^{\dagger n}E}{\partial\lambda_1(t_1)\partial\lambda_2(t_2)\ldots\partial\lambda_n(t_n)}$ can be obtained by calculating $P_1, P_2, \ldots, P_n$ iteratively, where

$$
\begin{aligned}
&P_3(j,t,\lambda_1(t_1),\lambda_2(t_2),\lambda_3(t_3)) \\
&\quad = \frac{\partial^{\dagger 3}h_j(t)}{\partial\lambda_1(t_1)\partial\lambda_2(t_2)\partial\lambda_3(t_3)} \qquad (33)
\end{aligned}
$$

$$
\begin{aligned}
&\bullet \\
&\bullet \\
&\bullet
\end{aligned}
$$

$$
P_n(j,t,\lambda_1(t_1),\ldots,\lambda_n(t_n)) = \frac{\partial^{\dagger n}h_j(t)}{\partial\lambda_1(t_1)\ldots\partial\lambda_n(t_n)} \qquad (34)
$$

Details of calculation of the n-th ordered derivatives can be found in [16].

4.3 Computational Complexity of Forward and Backward Propagation Schemes

Comparing the forward propagation scheme with the backward propagation scheme discussed in the previous sections, we can see that they are similar in form. However, the computation loads required in these two schemes are very different. Since $P_1(j,t,\lambda_1(t_1))$ is a function of parameter $\lambda_1(t_1)$, we need to calculate P_1 for every parameter. On the other hand, $\delta_1(j,t)$ in the backward propagation scheme does not depend on parameters, and therefore we only need to calculate a single time function $\delta_1(j,t)$ for a given node j, This result in, for calculation of the first ordered derivatives , a large computational load of the forward propagation scheme by a factor of the number of parameters involved. For this reason, the forward propagation algorithm is rarely used for the calculation of the first ordered derivatives.

The situations are different for calculation of the higher ordered derivatives. The forward propagation scheme requires lower computational load when

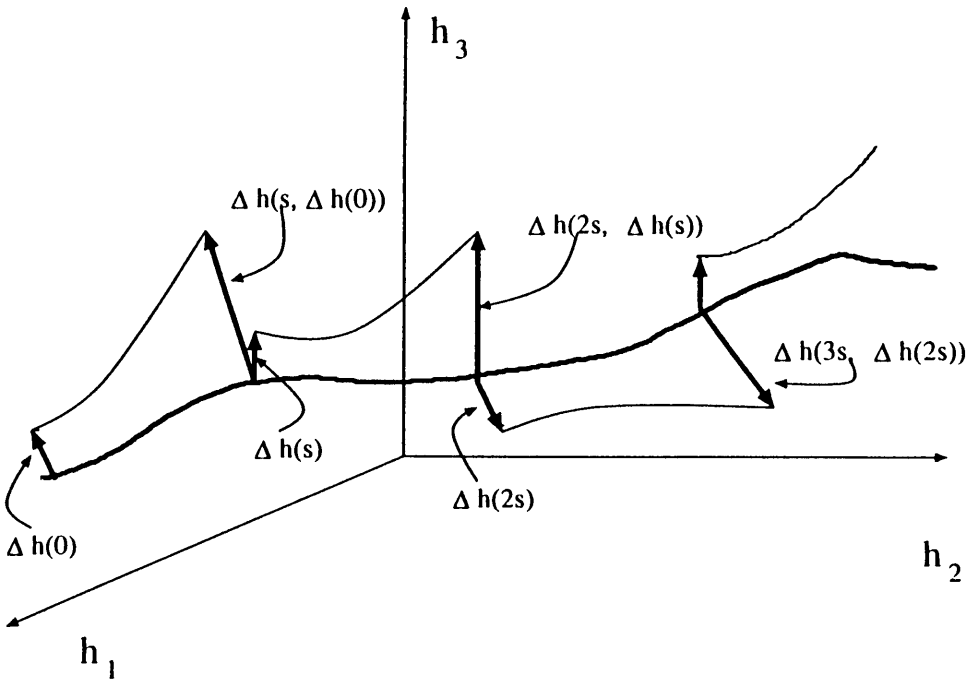

Fig.2 Calculation of maximum Lyapunov number

applied to calculation of the higher ordered derivatives. Let $|J|$ be the number of nodes, $|T|$ be the number of sampling instants and $|M|$ be the number of parameters. As explained in [15], [16], in the case of time-invariant parameter systems, the computational load of the n-th ordered derivatives by forward propagation is proportional to $|J|^2|T||M|^n$, whereas it is proportional to $|J|^{2n}|T|^n$ by backward propagation. In practical applications, for instance controller design, a large number of sampling times are usually involved. Therefore, when the higher ordered derivatives are required, it is highly recommened to use the forward propagation scheme.

5. Chaos Control Method

In this section, a new chaos control method which generate or eliminates chaotic phenomena by adjusting parameters in ULN whose signal is within bounds is proposed. The criterion function adopted for training parameters is maximum Lyapunov number which can be represented by the first order derivative of ULN. Therefore, parameter training by the gradient method can be carried out by using the second ordered derivative of ULN described in the previous sections, since maximum Lyapunov number itself is the first ordered derivative. The fundamental idea of the proposed method is that if maximum Lyapunov number of the dynamic system becomes greater tham zero by adjusting parameters, then the system behaves chaotically, otherwise, the system is stable.

The perpose of this section is to verify whether commonly used neural networks can generate chaos or not by the proposed method without any special mechanism which can control the chaotic behavior and also to demonstrate that the higher ordered derivatives can be effectively used to control the chaos.

Generally speaking, Lyapunov number is an index which describes the rate of expansion and contraction of the change of the dynamics caused by the initial variation. So Lyapunov number $\chi_{j,j\in J}$ of $|J|$ dimensional dynamics described by Eq.(35) can be expressed as follows.

$$h_j(t) = f_j(h(t-1)) \tag{35}$$

$$\chi_j = \lim_{L\to\infty} \frac{1}{L} \log_e |\sigma_j(L)| \tag{36}$$

$$DF_L = DF(h(L-1))DF(h(L-2))...DF(h(0)) \quad (37)$$

$$DF(h(t)) = \begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} & \cdots & \frac{\partial f_1}{\partial h_{|J|}} \\ \frac{\partial f_2}{\partial h_1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_{|J|}}{\partial h_1} & \frac{\partial f_{|J|}}{\partial h_2} & \cdots & \frac{\partial f_{|J|}}{\partial h_{|J|}} \end{bmatrix}$$

where $h(t) = (h_1(t), h_2(t), ..., h_{|J|}(t))$
$\sigma_j(L)$: eign value of matrix DF_L

Obviously, from that mentioned above, computing Lyapunov number for multiple-dimensional dynamics is very complicated since many matrix operations and eigen value calculation are necessary. Therefore, we adopted the following maximum Lyapunov number χ represented by Eq.(38) and Eq.(39).

$$\chi = \lim_{t\to\infty} \frac{1}{t} \log_e \frac{\|\Delta h(t, \Delta h(0))\|}{\|\Delta h(0)\|} \quad (38)$$

where $\Delta h(0) = (\Delta h_1(0), \Delta h_2(0), ..., \Delta h_{|J|}(0))$
: initial variation of node outputs
$\Delta h(t, \Delta h(0)) = (\Delta h_1(t, \Delta h(0)),$
$\Delta h_2(t, \Delta h(0)), ..., \Delta h_{|J|}(t, \Delta h(0)))$
: change of dynamics caused by the initial variation $\Delta h(0)$

By considering that $\Delta h(t, \Delta h(0))$ in Eq.(38) can be approximately calculated by using the following Taylor expansion

$$\Delta h_r(t, \Delta h(0)) = \sum_{r1\in J} \frac{\partial^\dagger h_r(t)}{\partial h_{r1}(0)} \Delta h_{r1}(0) \quad (39)$$

and also taking account of the overflow when calculating Eq.(39), the maximum Lyapunov number χ is approximately calculated in the following way

$$\chi = \lim_{L\to\infty} \frac{1}{Ls} \sum_{l=0}^{L-1} \log_e \frac{\sqrt{\sum_{r\in J} \Delta h_r(l+s, \Delta h(l))^2}}{\sqrt{\sum_{r\in J} \Delta h_r(l)^2}} \quad (40)$$

$$\Delta h_r(l+s, \Delta h(l)) = \sum_{r1\in J} \frac{\partial^\dagger h_r(l+s)}{\partial h_{r1}(l)} \Delta h_{r1}(l) \quad (41)$$

When calculating Eq.(40), the variation $\Delta h(l)$ is initialized every s sampling instants as shown in **Fig.2**.

5.1 Parameter Training with Gradient Method

The chaotic phenomena of ULNs can be controlled by adjusting the parameters for the maximum Lyapunov number represented by Eq.(40) to approach a postive or negative value. So the following criterion function is adopted

$$E = (\chi - \chi_o)^2 \quad (42)$$

where χ_0 is the target value of maximum Lyaunov number. Since Lyapunov number is represented by the first ordered derivative (Eq.(40) and Eq.(41)), the computation of the second ordered derivative described in the preceding sections is necessary when the above criterion function is used and the gradient method is adopted. Therefore, in order to minimize the criterion function, parameters of ULN are trained as follows

$$\lambda_m \leftarrow \lambda_m - \gamma \frac{\partial^\dagger E}{\partial \lambda_m} \quad (43)$$

$$\frac{\partial^\dagger E}{\partial \lambda_m} = 2(\chi - \chi_0) \frac{\partial^\dagger \chi}{\partial \lambda_m} \quad (44)$$

$$\frac{\partial^\dagger \chi}{\partial \lambda_m} = \lim_{L\to\infty} \frac{1}{Ls} \times \sum_{l=0}^{L-1} \frac{\sum_{r\in J} \Delta h_r(l+s, \Delta h(l)) \frac{\partial^\dagger \Delta h_r(l+s, \Delta h(l))}{\partial \lambda_m}}{\sum_{r\in J} \Delta h_r(l+s, \Delta h(l))^2} \quad (45)$$

$$\frac{\partial^\dagger \Delta h_r(l+s, \Delta h(l))}{\partial \lambda_m} = \sum_{r1\in J} \left[\frac{\partial^{\dagger 2} h_r(l+s)}{\partial h_{r1}(l) \partial \lambda_m} \Delta h_{r1}(l)\right] \quad (46)$$

where γ : learning coefficient
$\frac{\partial^\dagger h_r(l+s)}{\partial h_{r1}(l)}$, $\frac{\partial^{\dagger 2} h_r(l+s)}{\partial h_{r1}(l) \partial \lambda_m}$ in Eq.(45) and Eq.(46) can be calculated by using the higher ordered derivatives computation with forward propagation de-

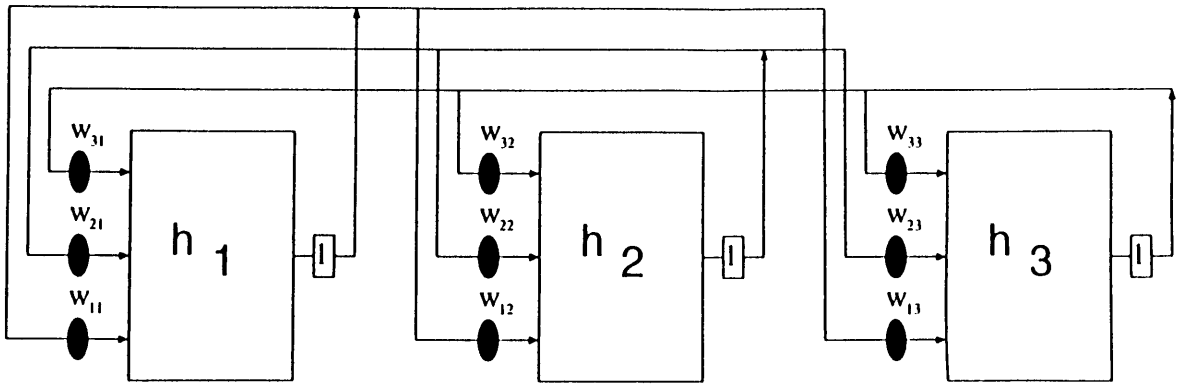

Fig.3 Recurrent Neural Network For Simulation

Table 1 Simulation Conditions

initial value of weights w_{ij} : random between	[-2,2]
initial value of slopes ϕ_j	1.0
initial value of threshold θ_j	0.0
learning coefficient of weights γ for w_{ij}	0.004
learning coefficient of slopes γ for ϕ_j	0.00001
learning coefficient of thresholds γ for θ_j	0.00001
range of random searching	0.25
target value of maximum Lyapunov number χ_0	3.0
sampling instant s	150
L in Eq.(39)	3
initial variation $\Delta h_1(l), \Delta h_2(l), \Delta h_3(l)$	0.01

scribed in section 4.

The reason why the forward propagation method instead of the backward propagation method is used is that the forward propagation scheme requires lower computational load when applied to calculation of the higher ordered derivatives described in section 4.3.

5.2 Local-Minimum Problem

The value of maximum Lyapunov number changes abruptly with small variations of parameters when the value of maximum Lyapunov number is positive because the criterion function including maximum Lyapunov number is not a monotonic function but a complicated one(**Fig.4**). Therefore,it is very easy to be stuck at a local minimum. So, a random search method is combined with the gradient method to globally optimize the parameters.

During the iteration of parameter training with the gradient method, the parameters will be changed randomly within a fixed range if the criterion function increases. The random searching will continue untill the criterion function decreases less than the one obtained by the last learning epoch of the gradient method. After that the learning with the gradient method resumes again using new parameters.

6. Computer Simulation

6.1 Sample Network for Simulations

A sample network for simulations is a fully-connected recurrent neural network with three nodes. And, the following sigmoid functions are adopted as network's node functions

$$h_j(t) = f_j(\alpha_j) = \frac{1.0}{1.0 + \exp(-\phi_j \alpha_j)} \tag{47}$$

$$\alpha_j = \sum_{i \in JF(j)} w_{ij} h_i(t-1) + \theta_j \qquad (48)$$

where
θ_j, ϕ_j : threshold and slope parameters of sigmoid function
w_{ij} : weight parameters from node i to node j

6.2 Simulation Results

6.2.1 Generation of chaotic phenomena

Target value of the maximum Lyapunov number is set to positive number three. After training the network by the combined gradient and random search method, learnig curve of maximum Lyapunov number, weight, threshold and slope parameters of sigmoid functions are obtained as shown in **Fig.4, 5, 6, 7**. Network dynamics is shown in **Fig.8, 9, 10, 11**. At learning epoch 5, dynamical trajectory shown in **Fig.8** converges to a fixed point where the maximum Lyapunov number is -0.859. At learning epoch 50, dynamical trajectory shown in **Fig.9** also converges with the maximum Lyapunov number being -0.038. At learning epoch 450, dynamical trajectory shown in **Fig.10** becomes chaotic as the maxium Lyapunov number is 0.215. The maximum Lyapunov number 0.359 at learning iteration 1150 is a proof that the dynamics shown in **Fig.11** behaves completely chaotically.

6.2.2 Die-out of Chaotic phenomena

Die-out of chaotic phenomena is realized by setting the value of maximum Lyapunov number to negative three. Learning curves of maximum Lyapunov number, weight, slope and threshold parameters of sigmoid functions are respectively shown in **Fig.12, 13, 14, 15**. As was stated in the subsection 5.2, the maximum Lyapunov number around or more than zero is very sensitive to parameter changes as shown in **Fig.4**. Therefore the value of maximum Lyapunov number can decrease very quickly as shown in **Fig.12** if target value χ_0 is assigned with a negative value.

Network dynamics at epoch 1 and epoch 2000 are respectively shown in **Fig.16** and **17**. The network dynamics shown in **Fig.16** is chaotic because its maximum Lyapunov number is equal to 0.213. But after training parameters 2000 times, the network has been completely stabilized while its maximum Lyapunov number decreased down to -2.434.

6.3 Discussion

From **Fig.4**, it is evident that as the value of the maximum Lyapunov number increases through learning, the absolute values of the weight and slope parameters tend to increase. The reason for this tendancy is as follows. It is clear from Eq.(23),(40) and (41), that $|\frac{\partial^\dagger h_r(l+s)}{\partial h_{r1}(l)}|$, in other words, $|\frac{\partial h_j(t)}{\partial h_i(t-D_{ij}(p))}|$ should be large enough for the value of the maximum Lyapunov number to increase. As the following equation hold,

$$\left|\frac{\partial h_j(t)}{\partial h_i(t-1)}\right| = |w_{ij}||\phi_j|(1-h_i(t-1))h_i(t-1) \qquad (49)$$

$|w_{ij}|$ and $|\phi_j|$ should be large for the system to behave chaotically.

7. Conclusion

Universal Learning Network (ULN) and its application to chaos control have been discussed. In regard of architectures and learning algorithm, a class of neural networks are unified in the framework of ULNs. In an architecture aspect, ULNs have higher degree of freedom : arbitrary continously differentiable functions in the nodes, multiple branches and arbitrary time delays. The ULN training algorithm, in either backward propagation scheme or forward propagation scheme, is equipped with a mechanism for calculating the higher ordered derivatives as a noble feature that has not been possesed by common neural networks. Therefore, ULNs are natural and furthest extensions of discrete-time recurrent nerual networks, which are useful for modeling and controling large scale complicated systems such as industrial plants, economic, social and life phenomena, and the ULNs can be applied widely to many kinds of applications.

In this paper, a new chaos control method is also proposed which can generate or eliminate chaotic phenomena by controlling the maximum Lyapunov number of dynamic systems that can be represented by ULNs. A new method based on the gradient and random search method is used to minimize the criterion function which is the difference between the desired maximum Lyapunov number and its actual value. Second ordered derivatives of ULN can be effectively used to train the parameters of the network in the gradient learning method.

Simulation results show that the generation and elimination of chaotic phenomena of ULN are easily realized by using the proposed method. How-

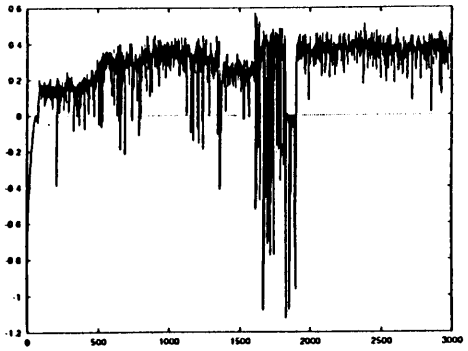

Fig.4 maximum Lyapunov number

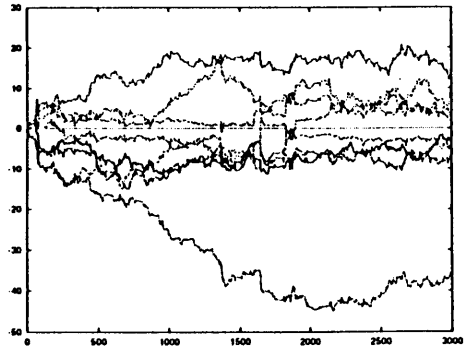

Fig.5 weight parameters

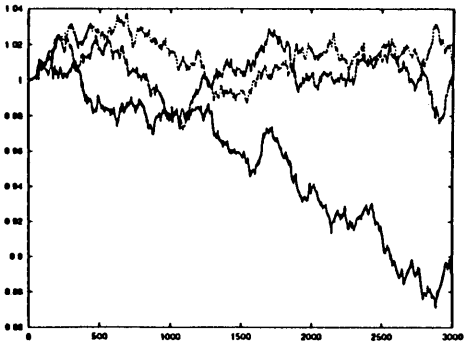

Fig.6 slope parameters

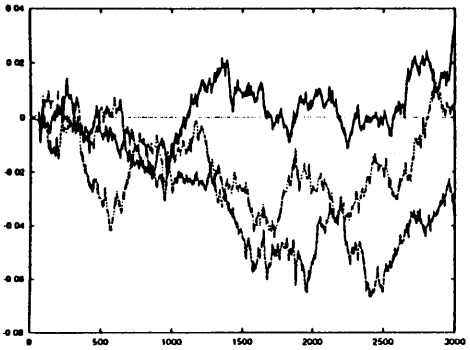

Fig.7 threshold parameters

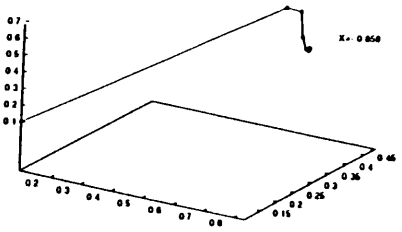

Fig.8 dynamics at epoch 5

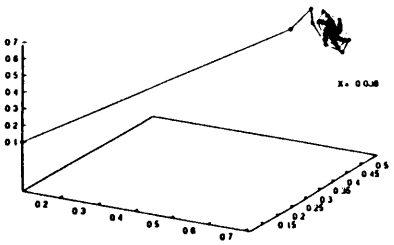

Fig.9 dynamics at epoch 50

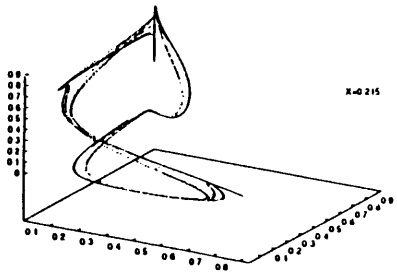

Fig.10 dynamics at epoch 450

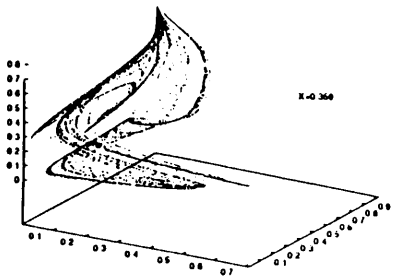

Fig.11 dynamics at epoch 1150

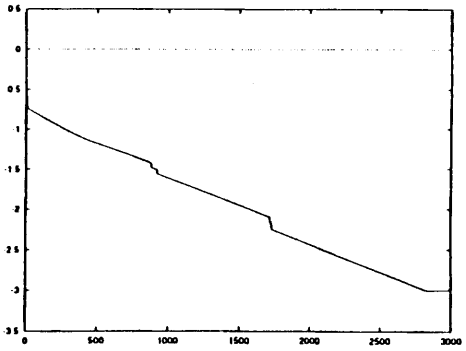

Fig.12 maximum Lyapunov number

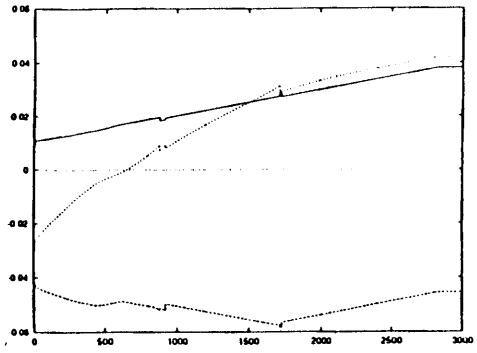

Fig.15 threshold parameters

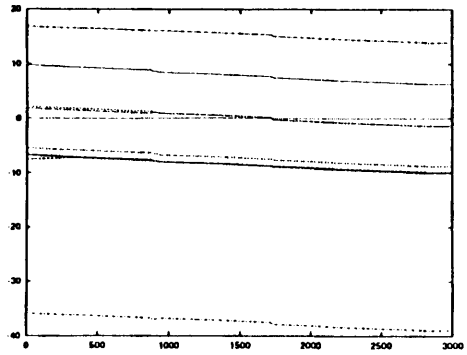

Fig.13 weight parameters

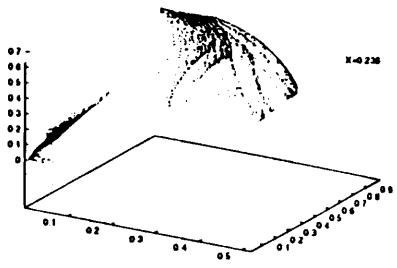

Fig.16 dynamics at epoch 1

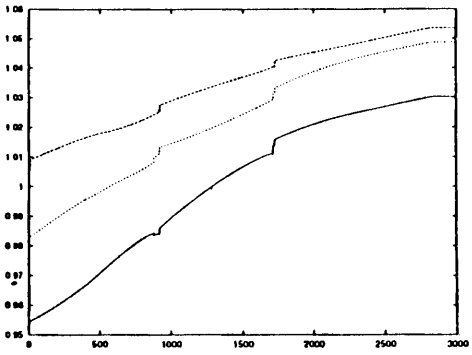

Fig.14 slope parameters

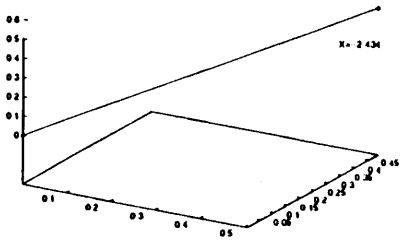

Fig.17 dynamics at epoch 2000

ever,the optimization of parameters is fairly difficult when the value of the maximum Lyapunov number is positive. The work that should be done in a future is to find a more powerful search method and to control the system's sensitivity and stablity efficiently.

References

1) D. E. Rumelhart, G. E. Hinton and R. J. Williams: **Learning Representation by Back-Propagating Errors**, *Nature*, 323,pp.533-536(1986)
2) F. Rosenblat,: **The Perceptron, a Probabilistic Model for Information Storage and Organization in the Brain**, *Psychological Review*, 65,pp.386-408(1958)
3) R. J. Williams and D. Zipser: **A Learning Algorithm for continually Running Fully Recurrent Neural Networks**, *Neural Computation*, 1,No.2,pp.270-280(1989)
4) R. J. Williams and D. Zipser: **Gradient-Based Learning Algorithms for Recurrent Connectionist Networks**, *College of Computer Science Technical Report, Northeastern University*, NU-CCS-90-9,pp.1-45(1990)
5) T. Poggio and F. Girosi: **Networks for Approximation and Learning**, *Proc.IEEE*, 78,9,pp.1481-1497(1990)
6) M. M. Gupta and J. Qi: **On fuzzy neuron mod-**

els, *Proc of Int. Joint Conf. on Neural Networks,* Vol.2,pp.431-436(1991)

7) D. T. Lin, J. E. Dayhoff and P. A. Ligomenides: **Trajectory Production with the Adaptive Time-Delay Neural Network**, *Neural Networks,* Vol.8,No.3,pp.447-461(1995)
8) W. L. Buntine and A. S. Weigend: **Computin Second Derivatives in Feed-Forward Network: A Riview**, *IEEE Trans. on Neural Networks,* vol.5,No.3(1994)
9) M. Han, K. Hirasawa, M. ohbayashi and M. Fujita: **Modeling Dynamic Systems Using Universal Learning Network**, *Proc of IEEE Conference on System, Man and Cybernetics,* pp.1172-1177(1996)
10) K. Aihara, T. Takabe and M. Toyada: **Chaotic Neural Networks** *Phys.Lett.A*, 144,6,7,pp333-340(1990)
11) Luonan. Chen and K. Aihara: **Chaotic Simulated by a Neural Network Model with Transient Chaos,** *Neural Network,* vol.8,No.6,pp915-930(1995)
12) R. J. Williams and D. Zipser: **A Learning Algorithm for continually Running Fully Recurrent Neural Networks**, *Neural Computation,* 1,No.2,pp.270-280(1989)
13) Kensuke. Ikeda and Kenji. Matsumoto: **High Dimensional Chaotic Behavior in Systems with Time-Delay Feedback**, *Physica 29D,* pp.223-235(1987)
14) P. Werbos: **Beyond regression: New Tools for Prediction and Analysis in the Behavior Science**, *Ph.D. dissertation, Harvard University,* (1974)
15) K. Hirasawa, M. Ohbayashi and J. Murata: **Universal Learning Network and Computation of its Higher Order Derivatives**, *Proc. of 1995 IEEE International Conference on Neural Networks,* pp.1273-1277(1995)
16) K. Hirasawa, M. Ohbayashi, M. koga and M. Harada: **Forward Propagation Universal Learning Network**, *Proc of 1996 IEEE International Conference on Neural Networks,* pp.353-358(1996)
17) E. Ott, C. Grebogi and J. A. Yorke:**Controlling chaos**, *Phys.Lett,* 64,(1990)