15th Triennial World Congress, Barcelona, Spain

A STATISTICAL APPROXIMATION LEARNING METHOD FOR SIMULTANEOUS RECURRENT NETWORKS

Masao Sakai * **Noriyasu Homma** ** **Kenichi Abe** *

* *Department of Electrical and Communication Engineering, Graduate School of Engineering, Tohoku University Aoba 05, Aramaki, Aoba-ku, Sendai, 980-8579, Japan E-mail : {sakai, abe}@abe.ecei.tohoku.ac.jp*
** *Department of Radiological Technology College of Medical Sciences, Tohoku University 2-1 Seiryo-machi, Aoba-ku, Sendai, 980-8575, Japan Email: homma@abe.ecei.tohoku.ac.jp*

Abstract: In this paper, a statistical approximation learning (SAL) method is proposed for a new type of neural networks, simultaneous recurrent networks (SRNs). The SRNs have the capability to approximate non-smooth functions which cannot be approximated by using conventional multi-layer perceptrons (MLPs). However, the most of the learning methods for the SRNs are computationally expensive due to their inherent recursive calculations. To solve this problem, a novel approximation learning method is proposed by using a statistical relation between the time-series of the network outputs and the network configuration parameters. Simulation results show that the proposed method can learn a strongly nonlinear function efficiently. *Copyright © 2002 IFAC*

Keywords: Backpropagation, dynamic modelling, learning algorithms, model approximation, neural networks and statistical approximation

1. INTRODUCTION

In the field of supervised learning, the most popular form of the feedforward neural networks, the multi-layer perceptrons (MLPs) (Andrew Barron, 1994), have been proven to approximate smooth functions very well, then many application problems use the MLPs as a model for identifying and controlling nonlinear complex dynamic systems (Narendra and Parthasarathy, 1990). However there are many practical and difficult applications, e.g., a brain-like intelligent control and planning, where the functions to be approximated are not smooth. In these cases, the MLPs cannot approximate the non-smooth functions accurately(Werbos, 1994).

In general, since the recurrent neural networks have some brain-like complex feedback connections, they can provide very complex dynamics. Indeed, in the state space of the dynamic neural system, using the *basin* which separates the initial state vectors from each other according to the corresponding final state vectors, a new type of neural networks, simultaneous recurrent networks (SRNs), has the capability to approximate non-smooth functions(Werbos, 1995).

There are several learning methods to train the SRNs. However, the most of these methods including the well-known backpropagation through time (BPTT) algorithm are computationally expensive due to calculation of the final state(Werbos, 1994). Also the recursive calculation of the gradients diverges when SRNs learns the strongly nonlinear function by these methods. To solve these problems, an approximation method, called the *truncation* method, has been proposed(Fausett, 1994). In this method, the recursive

terms are simply truncated (assumed to be 0). Therefore, this non-recursive method often leads to wrong learning directions.

In this paper, to improve the approximation accuracy without much increase of the computational time, a novel approximation learning method, a statistical approximation learning (SAL) method, is proposed. The proposed method can approximate the gradient by using a statistical relation between the networks dynamics and the parameters of the SRNs. Simulation results show that the proposed SAL method can efficiently learn a strongly nonlinear function which cannot be learned by the conventional methods. Also, due to the non-recursive formulation of the approximation, it is shown that the SAL method can provide a practical computational time compared with that of the conventional BPTT method.

2. SIMULTANEOUS RECURRENT NETWORKS

Fig.1 shows the simultaneous recurrent networks (SRNs). For this SRN model, the following variables are introduced:

$\boldsymbol{u}$	:	The external neural input vector, $\boldsymbol{u} = [u_1\ u_2\ \cdots\ u_r]^\top$, r is the number of external inputs.
s_i, x_i	:	The input and output of the ith neuron in the hidden layer. $i = 1, 2, \cdots, n$, n is the number of neurons in the hidden layer.
y_k	:	The output of the kth neuron in the output layer. $k = 1, 2, \cdots, m$, m is the number of neurons in the output layer.
$\overline{\boldsymbol{y}}$	:	The neural output vector, $\overline{\boldsymbol{y}} = [\overline{y}_1\ \overline{y}_2\ \cdots\ \overline{y}_m]^\top$,
v_{ki}	:	The weight connecting x_i to y_k.
$w_{i\nu}$	:	The weight connecting u_ν to x_i.
$\mu_{i\kappa}$	:	The feedback weight connecting y_κ to x_i.
θ_k	:	The bias of the kth neuron in the output layer.
ϕ_i	:	The bias of the ith neuron in the hidden layer.

Let h be discrete time, $h = 1, 2, \cdots$. The neural inputs and outputs at time h in the hidden layer, $s_i(h)$ and $x_i(h)$, $i = 1, 2, \cdots, n$, are defined as

$$s_i(h) = \sum_{\nu=1}^{r} w_{i\nu} u_\nu + \sum_{\kappa=1}^{m} \mu_{i\kappa} y_\kappa(h-1) + \phi_i, \quad (1)$$

$$x_i(h) = \frac{1}{1 + \exp\left(-\alpha s_i(h)\right)}, \quad (2)$$

where, α is a gain coefficient of the sigmoid function. The neural outputs in the output layer, $y_k(h)$, $k = 1, 2, \cdots, m$, are defined as

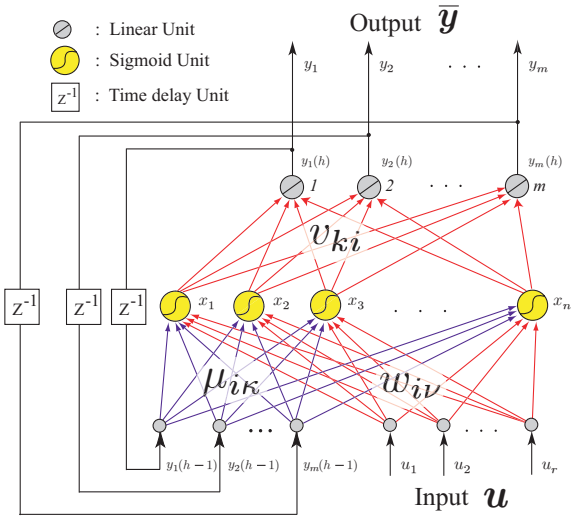

Fig. 1. Simultaneous recurrent networks (SRNs).

$$y_k(h) = \sum_{i=1}^{n} v_{k\,i} x_i(h) + \theta_k\,. \quad (3)$$

The network outputs of the SRN are given as

$$\overline{y}_k = \lim_{H\to\infty} y_k(H), \quad (4)$$

where, H is the number of iterations. In this paper let H be 10 instead of ∞. Note that the conventional MLPs are only a special case of the SRNs with a configurations as $H = 1$ and $y_\kappa(0) = 0$.

3. LEARNING METHODS FOR SRN

Supervised learning for the neural networks is the task of learning a nonlinear function in which several sample inputs and outputs of the function are given. Letting $\left\{ \left(\boldsymbol{u}^{(t)}, \boldsymbol{y}^{d(t)} \middle| t = 1, 2, \cdots, T \right) \right\}$ be the set of desired input-output pairs, the changes of parameters, δp, are given by the learning methods based on a backpropagation algorithm as (Rumelhart *et al.*, 1986)

$$\begin{aligned} \delta p &= -\eta \frac{\partial e}{\partial p}, \\ &= -\eta \sum_{t=1}^{T} \sum_{k=1}^{m} \left(\overline{y}_k^{(t)} - y_k^{d(t)} \right) \frac{\partial \overline{y}_k^{(t)}}{\partial p}, \end{aligned} \quad (5)$$

where $e = \sum_{t=1}^{T} \sum_{k=1}^{m} \left(\overline{y}_k^{(t)} - y_k^{d(t)} \right)^2 / 2$ is a square error and η is a positive coefficient. If the gradients $\partial \overline{y}_k^{(t)} / \partial p$ are obtained, the change of parameters δp can be calculated. In the following, the calculations of the $\partial \overline{y}_k / \partial p$ by conventional methods, i.e., the backpropagation through time (BPTT) method and the *truncation* method are described briefly.

3.1 *Backpropagation through time method*

The backpropagation through time (BPTT) method is a general method which calculates the gradients exactly. From (3) and (4), the gradients $\partial \overline{y}_k/\partial p$ and $\partial y_k(h)/\partial p$ are given as

$$\frac{\partial \overline{y}_k}{\partial p} = \lim_{H\to\infty} \frac{\partial y_k(H)}{\partial p}, \tag{6}$$

$$\frac{\partial y_k(h)}{\partial p} = \Delta_p(\theta_k) + \Delta_p(v_{k\ell}) \cdot x_\ell(h) + \sum_{i=1}^{n} v_{ki} \frac{\partial x_i(h)}{\partial p}, \tag{7}$$

where $\Delta_p(v)$ is defined by

$$\Delta_p(v) = \begin{cases} 1, & (p = v), \\ 0, & (p \neq v). \end{cases} \tag{8}$$

By (1) and (2), $\partial x_i(h)/\partial p$ and $\partial s_i(h)/\partial p$ are calculated as

$$\frac{\partial x_i(h)}{\partial p} = \alpha x_i(h)\left(1 - x_i(h)\right) \frac{\partial s_i(h)}{\partial p}, \tag{9}$$

$$\frac{\partial s_i(h)}{\partial p} = \Delta_p(w_{i\nu}) \cdot u_\nu + \Delta_p(\mu_{i\ell}) \cdot y_\ell(h-1) + \sum_{\kappa=1}^{m} \mu_{i\kappa} \frac{\partial y_\kappa(h-1)}{\partial p} + \Delta_p(\phi_i). \tag{10}$$

Here letting $\partial y_\kappa(0)/\partial p$ be 0 as initial values, $\partial s_i(1)/\partial p$ are given by (10), then $\partial y_k(1)/\partial p$ are calculated by (7) $\sim$ (9). Finally the gradients $\partial \overline{y}_k/\partial p$ ($\equiv \partial y_i(H)/\partial p$) are obtained by calculating $\partial y_k(h)/\partial p$ recursively.

3.2 *Truncation method*

The *truncation* method is probably the most popular method used to adapt SRNs even though the people who use it mostly just call it ordinary backpropagation. To calculate the gradients $\partial \overline{y}_k/\partial p$, the method uses only one simple pass of backpropagation through the last iteration of the model by truncating $\partial y_k(H-1)/\partial p = 0$. Therefore the method is the simplest approximation and the least expensive method.

4. STATISTICAL APPROXIMATION LEARNING METHOD

4.1 *Basic strategy for a statistical approximation*

The gradients $\partial \overline{y}_k/\partial p$ calculated by the *truncation* method are not accurate. Therefore it often fails to learn the strongly nonlinear models. To improve the accuracy, a statistical approximation learning (SAL) method is proposed. In the proposed method, the truncation, $\partial y_k(H-1)/\partial p = 0$ are redefined using a statistical approximation. That is, the recursive gradients $\partial x_i(h)/\partial p$ in (7) are approximated by using the expectation of x_i, $E[x_i]$, as

$$\frac{\partial x_i(h)}{\partial p} \approx \frac{\partial E[x_i]}{\partial p}. \tag{11}$$

Then, from (7)

$$\frac{\partial y_k(H-1)}{\partial p} \approx \Delta_p(\theta_k) + \Delta_p(v_{k\ell}) \cdot x_\ell(H-1) + \sum_{i=1}^{n} v_{ki} \frac{\partial E[x_i]}{\partial p}. \tag{12}$$

To calculate the gradients of the expectation, $\partial E[x_i]/\partial p$, since the SRNs are only a special case of the fully connected recurrent networks, an equivalent notation of the SRNs to the fully connected recurrent networks is developed. In the fully connected recurrent networks, the expectation $E[x_i]$ is represented by a function of a key parameter $\overline{\sigma^2}$, which implies a variance of the inputs s_i in (1)(Sakai *et al.*, 2001). Using the equivalent notation and the statistical relation between the expectation $E[x_i]$ and the key parameter $\overline{\sigma^2}$, the gradients $\partial E[x_i]/\partial p$ can be approximated by

$$\frac{\partial E[x_i]}{\partial p} \approx \frac{\partial E[x_i]}{\partial \overline{\sigma^2}} \cdot \frac{\partial \overline{\sigma^2}}{\partial p}. \tag{13}$$

Let $\omega_{ji} \equiv \sum_{\kappa=1}^{m} \mu_{j\kappa} v_{\kappa i}$, and $\psi_j \equiv \sum_{\nu=1}^{r} w_{j\nu} u_\nu + \sum_{\kappa=1}^{m} \mu_{j\kappa}\theta_k + \phi_j$, then the SRNs defined by (1) $\sim$ (3) are equivalent to networks contained fully connected recurrent networks given as

$$y_k(h) = \sum_{j=1}^{n} v_{kj} x_j(h) + \theta_k, \tag{14}$$

$$x_j(h) = \frac{1}{1 + \exp(-\alpha s_j(h))}, \tag{15}$$

$$s_j(h) = \sum_{i=1}^{n} \omega_{ji} x_i(h-1) + \psi_j. \tag{16}$$

To apply the statistical relation to the equivalent SRNs, let ω_{ji} and ψ_j be uniformly distributed random variables whose expected value is zero. The inputs are given by $s_j = \sum_i s_{ji} + \psi_j$, where $s_{ji} \equiv \omega_{ji} x_i$. Supposing that x_i, ω_{ji} and ψ_j are independent of each other, and that the outputs x_i are uniformly random numbers between 0 to 1 when networks dynamics are chaotic(Sakai *et al.*, 2001). Then the expectations of s_{ji} and ψ_j are given by $E[s_{ji}] = E[w_{ji}] \cdot E[x_i] = 0$ and $E[\psi_j] = 0$. The variances are respectively given by $\sigma^2(s_{ji}) = E[\omega_{ji}^2] \cdot E[x_i^2]$ and $\sigma^2(\psi_j) = E[\psi_j^2]$, where $\sigma^2(z)$ denotes the variance of a variable z. Using the above supposition, $E[x_i^2]$ is calculated as $E[x_i^2] = 1/3$. The expectations converge to the ensemble averages as $n \to \infty$, thus $E[\omega_{ji}^2] = \sum_{j,i} \omega_{ji}^2/n^2$ and $E[\psi_j^2] = \sum_j \psi_j^2/n$. Therefore by the low of large numbers, a variance $\overline{\sigma^2}$ of the inputs s_i is calculated by

$$\overline{\sigma^2} \approx n\sigma^2(s_{ji}) + \sigma^2(\psi_j),$$
$$= \frac{1}{3n}\sum_{j=1}^{n}\sum_{i=1}^{n}\omega_{ji}^2 + \frac{1}{n}\sum_{j=1}^{n}\psi_j^2. \qquad (17)$$

Here $\partial\overline{\sigma^2}/\partial v_{r\ell}$ is calculated by the definitions of ω_{ji} and ψ_j as

$$\frac{\partial\overline{\sigma^2}}{\partial v_{r\ell}} = \frac{2}{3n}\sum_{j=1}^{n}\sum_{i=1}^{n}\omega_{ji}\frac{\partial\omega_{ji}}{\partial v_{r\ell}} + \frac{2}{n}\sum_{j=1}^{n}\psi_j\frac{\partial\psi_j}{\partial v_{r\ell}},$$
$$= \frac{2}{3n}\sum_{j=1}^{n}\omega_{j\ell}\mu_{jr}. \qquad (18)$$

Also $\partial\overline{\sigma^2}/\partial p,\ p \in \{\theta_r, w_{r\ell}, \mu_{r\ell}, \phi_r\}$ can be calculated similarly. Therefore, if the gradients $\partial E[x_i]/\partial\overline{\sigma^2}$ are calculated, then the target gradients in (13), $\partial E[x_i]/\partial p$ can be calculated.

4.2 *Calculation of* $\partial E[x_i]/\partial\overline{\sigma^2}$

From (2), the sigmoid function is given as a following power series representation(Sakai *et al.*, 2001).

$$x_i \approx \begin{cases} 1, & (s_i > \epsilon/\alpha), \\ \displaystyle\sum_{\tau=0}^{M_1} F(\tau)\cdot(\alpha s_i)^\tau, & (|s_i| < \epsilon/\alpha), \\ 0, & (s_i < -\epsilon/\alpha), \end{cases} \qquad (19)$$

$$F(\tau) \equiv \begin{cases} 1/2, & (\tau = 0), \\ \displaystyle\sum_{k=1}^{\tau}\frac{(-1)^{k+1}}{2k!}\cdot F(\tau-k), & (\tau > 0), \end{cases} \qquad (20)$$

where M_1 is a suitable natural number and ϵ is a positive constant. The probability density $g(s_i, 0, \overline{\sigma^2})$ of the expectation of the input s_i can be given as a power series representation(Sakai *et al.*, 2001).

$$g\left(s_i, 0, \overline{\sigma^2}\right) = \frac{1}{\sqrt{2\pi\overline{\sigma^2}}}\exp\left(-\frac{s_i^2}{2\overline{\sigma^2}}\right),$$

$$\approx \begin{cases} \displaystyle\sum_{\tau=0}^{M_2} R(\tau)\left(\frac{1}{\overline{\sigma^2}}\right)^{1/2}\left(\frac{s_i^2}{\overline{\sigma^2}}\right)^\tau, & \left(|s_i| < \beta\sqrt{\overline{\sigma^2}}\right), \\ 0, & (\text{ otherwise}), \end{cases} \qquad (21)$$

$$R(\tau) \equiv \frac{(-1)^\tau}{\sqrt{2\pi}\cdot 2^\tau\cdot\tau!}, \qquad (22)$$

where M_2 is a suitable natural number and β is a positive constant. Supposing $M_1 = 2M_2$ to avoid much complicated representation, $E[x_i]$ can be given by a power series representation as

$$E[x_i] = \int_{-\infty}^{\infty} x_i\cdot g(s_i, 0, \overline{\sigma^2})\ ds_i$$

$$\approx \begin{cases} \displaystyle\sum_{\substack{0\le\tau\le M_2\\ 0\le k\le\tau}} Q(\tau,k)\beta^{2\tau+1}\left(\alpha^2\overline{\sigma^2}\right)^k \\ \quad + \displaystyle\sum_{\substack{M_2+1\le\tau\le 2M_2\\ \tau-M_2\le k\le M_2}} Q(\tau,k)\beta^{2\tau+1}\left(\alpha^2\overline{\sigma^2}\right)^k, \\ \qquad\qquad \left(\alpha^2\overline{\sigma^2} < \dfrac{\epsilon^2}{\beta^2}\right), \\ \displaystyle\sum_{\substack{0\le\tau\le M_2\\ 0\le k\le\tau}} Q(\tau,k)\epsilon^{2\tau+1}\left(\alpha^2\overline{\sigma^2}\right)^{(k-\tau)-\frac{1}{2}} \\ \quad + \displaystyle\sum_{\substack{M_2+1\le\tau\le 2M_2\\ \tau-M_2\le k\le M_2}} Q(\tau,k)\epsilon^{2\tau+1}\left(\alpha^2\overline{\sigma^2}\right)^{(k-\tau)-\frac{1}{2}} \\ \quad + \displaystyle\sum_{\tau=0}^{M_2}\frac{R(\tau)}{(2\tau+1)}\left\{\epsilon^{2\tau+1} - \left(\frac{\epsilon^2}{\alpha^2\overline{\sigma^2}}\right)^{\tau+\frac{1}{2}}\right\}, \\ \qquad\qquad \left(\alpha^2\overline{\sigma^2} > \dfrac{\epsilon^2}{\beta^2}\right), \end{cases} \qquad (23)$$

$$Q(\tau,k) \equiv \gamma\frac{2\cdot F(2k)\cdot R(\tau-k)}{(2\tau+1)}, \qquad (24)$$

where γ is a positive constant. From the above arrangements the $\partial E[x_i]/\partial\overline{\sigma^2}$ can be calculated by (23) and (24) easily. Thus $\partial y_k(H-1)/\partial p$ in (12) can be calculated by (13) and (18). Finally, using the truncation algorithm and the $\partial y_k(H-1)/\partial p$ as the initial values, the gradients $\partial\overline{y}_k/\partial p$ can be calculated.

5. SIMULATION RESULTS

The three neural learning methods, the multi-layer perceptrons (MLPs) trained by the backpropagation (BP) method which is equivalent to the *truncation* method, the simultaneous recurrent networks (SRNs) trained by the backpropagation through time (BPTT) method and the *truncation* method, have been tested on a design task which requires the networks to learn a following *non-smooth* function (Fig.2).

$$y^d = \begin{cases} 1, & (u_1-0.5)(u_2-0.5) > 0\ , \\ 0.5, & (u_1-0.5)(u_2-0.5) = 0\ , \\ 0, & (u_1-0.5)(u_2-0.5) < 0\ . \end{cases} \qquad (25)$$

In this task, the networks with $m = 1, n = 30, r = 2, \alpha = 10$ were employed, and the constants and the coefficient were decided experimentally: $M_2 = 5$, $\epsilon = 2.4$, $\beta = 2$, $\gamma = 1.2$ and $\eta = 0.0002$. The connecting weights of the networks were initialized randomly, then changed by the learning methods. The number of learning iteration was 5000 and the number of training data was $T = 121$ (11×11 mesh data in $[u_1\ u_2]$ space, $u_1, u_2 \in \{0, 0.1, 0.2, \cdots, 1.0\}$).

The solid curve in Fig.3 shows the simulation result using the MLP with the BP method, and the input-output function of the MLP trained by the BP method

(after 5000 learning iterations) is shown in Fig. 4. Note that the MLP with the BP method cannot learn the target model accurately: the non-smooth parts of the target model are approximated by the smooth functions.

The dashed curve in Fig.3 shows the simulation result using the SRN with the BPTT method, and the input-output function of the SRN trained by the BPTT method is shown in Fig. 5. Note that the SRN with the BPTT method fails to learn the target model because the recursive calculations of the gradients diverged.

The dotted curve in Fig.3 shows the simulation result using the SRN with the *truncation* method, and the input-output function of the SRN trained by the *truncation* method is shown in Fig. 6. Note that the SRN with the *truncation* method can learn the target model better than the MLP with BP method. However, in the other simulation trials, the networks with different values of the initial states and connecting weights often fail to learn the target model. The typical failure is shown in Fig.7 (the dotted curve) and Fig.8 (the trained input-output function). Note that the SRN trained by the *truncation* method converges to the unsuitable local optimal solution. This is because of the inexact approximation of the *truncation* method.

On the other hand, the solid curve in Fig.7 shows the simulation result using the SRN with the SAL method, where the initial network states and parameters of the network are taken to be equal to those of SRN giving the input-output function in Fig.8 trained by the *truncation* method. The trained input-output function is shown in Fig. 9. As compared with Figs. 4–6 and Fig. 9, note that the SRN with the SAL method can learn the target more accurately than the three conventional methods.

Also, Table 1 lists the average error over 32 trials and CPU-TIMEs to calculate the gradients $\partial \overline{y}_k / \partial p$ in the similar simulations where the number of learning iterations was 10,000 and $n = 50$. In this case, the BPTT method cannot finish to train the SRN within a practical computational cost. Note that, as compared with the average errors over 32 trials, the average error of the SAL method, 6.98, is less than the average error, 12.07, by the *truncation* method under the same condition. In addition, for 4 trials out of 32 trials, the SRN trained by the *truncation* method fails to get a practical solution within 10,000 iterations. On the other hand, for all the 32 trials, the SAL method always leads to a good solution illustrated such as in Fig. 9. In brief, the accuracy of the SAL method is better than that of the *truncation* method. Also note that the SAL method can provide a practical computational time compared with that of the BPTT method.

Table 1. Average error over 32 trials and CPU-TIME

Learning method	Average error	CPU-TIME [sec.]
The MLP with the BP method	13.10	0.11
The SRN with the *truncation* method	12.07 (4*)	3.56
The SRN with the SAL method	6.98 (0*)	8.42
The SRN with the BPTT method	———	32.64

$*$ The number of the convergence to a local optimal solution

6. CONCLUSIONS

In this paper, a statistical approximation learning (SAL) method for the simultaneous recurrent networks (SRNs) has been proposed. In a simulation study, the proposed SAL method was tested. This simulation results showed that the proposed SAL method can efficiently learn the target model consisting of strongly nonlinear and non-smooth functions which cannot be learned by the conventional methods. Also, due to the non-recursive formulation of the approximation, it was shown that the SAL method can provide a practical computational time compared with that of the conventional BPTT method. Further applications using this method are in progress.

7. REFERENCES

Barron, A. (1994). Asymptotically optimal functional estimation by minimum complexity criteria. In: *Proc. of 1994 IEEE International Symposium of Information Theory*, IEEE, New York.

Fausett, L. (1994). *Fundamentals of Neural Networks: architectures, algorithms and applications.* Prentice Hall.

Narendra, K.S. and K. Parthasarathy (1990). Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks* **1**, 4–27.

Rumelhart, D.E., G.E. Hinton and R.J. Williams (1986). Learning representations by backpropagating errors. *Nature* **323**, 533–536.

Sakai, M., N. Honma and K. Abe (2001). Chaos control by stochastic analysis on recurrent neural networks. In: *Proc. of AROB 6th 2001*. Tokyo. **2**, 478–153.

Werbos, P.J (1994). *The Roots of Backpropagation.* John Wiley & Sons. New York.

Werbos, P.J. (1995). Optimization methods for brain-like intelligent control Organization of Behavior. In: *Proc. of the 34th Conference on Decision and Control 1995, IEEE Press.*, 579–584.

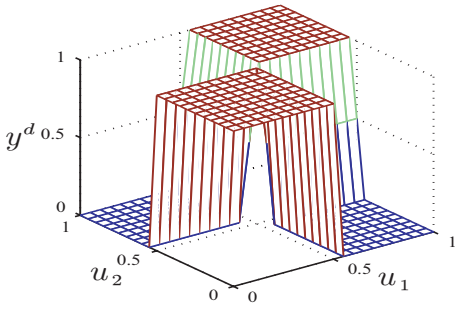

Fig. 2. The input-output function of the target model given in (25).

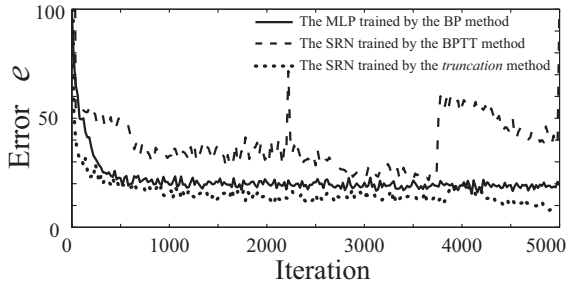

Fig. 3. The error e as functions of learning iteration by the three conventional neural learning methods, the MLP trained by the BP method (solid curve), the SRN trained by the BPTT method (dashed curve), and the SRN trained by the *truncation* method (dotted curve).

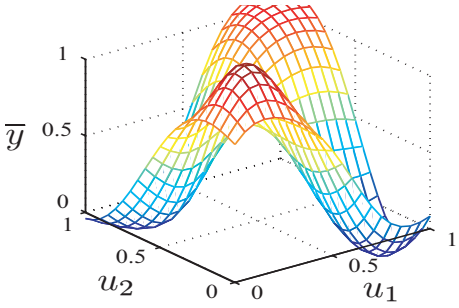

Fig. 4. The input-output function of the MLP trained by the BP method after 5000 learning iterations.

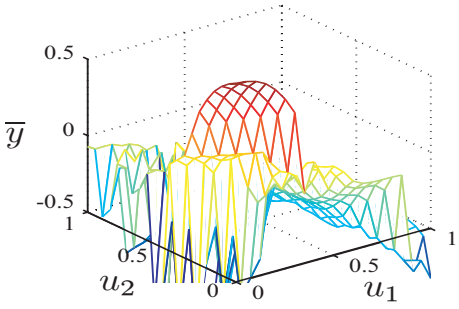

Fig. 5. The input-output function of the SRN trained by the BPTT method after 5000 learning iterations.

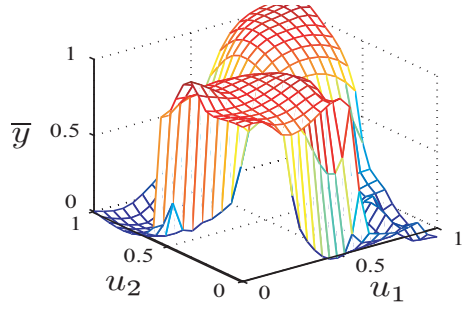

Fig. 6. The input-output function of the SRN trained by the *truncation* method after 5000 learning iterations (a successful result).

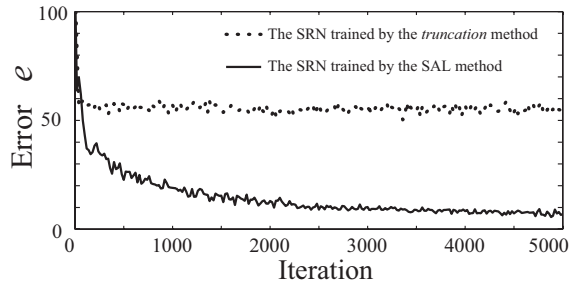

Fig. 7. The error e as functions of learning iteration for the SRN. The dotted curve shows the typical failure by the *truncation* method, while the solid curve shows the result by the proposed method, the statistical approximation learning (SAL) method.

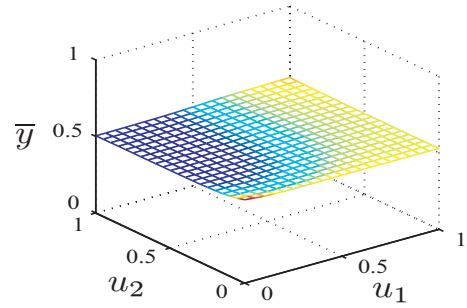

Fig. 8. The input-output function of the SRN trained by the *truncation* method after 5000 learning iterations (a fail result).

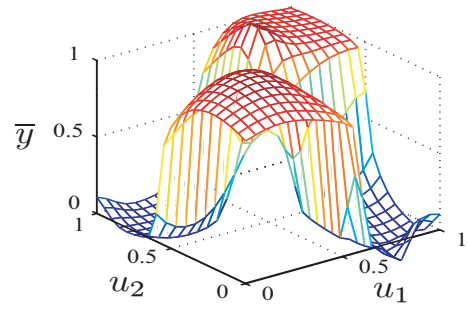

Fig. 9. The input-output function of the SRN trained by the SAL method after 5000 learning iterations.