A Fast Incremental Learning Algorithm of RBF Networks with Long-Term Memory

Keisuke Okamoto, Seiichi Ozawa, and Shigeo Abe
Graduate School of Science and Technology, Kobe University
1-1 Rokko-dai, Nada, Kobe 657-8501, JAPAN
Email: okamoto@frenchblue.scitec.kobe-u.ac.jp
{ozawa, abe}@eedept.kobe-u.ac.jp

***Abstract*— To avoid the catastrophic interference in incremental learning, we have proposed Resource Allocating Network with Long Term Memory (RAN-LTM). In RAN-LTM, not only a new training sample but also some memory items stored in Long-Term Memory are trained based on a gradient descent algorithm. In general, the gradient descent algorithm is usually slow and can be easily fallen into local minima. To solve these problems, we propose a fast incremental learning algorithm of RAN-LTM, in which its centers are not trained but selected based on output errors. This model does not need so much memory capacity and it also realizes robust incremental learning ability. To verify these characteristics of RAN-LTM, we apply it to two function approximation problems: one-dimensional function approximation and prediction of Mackey-Glass time series. From the experimental results, it is verified that the proposed RAN-LTM can learn fast and accurately without large main memory unless incremental learning is conducted over a long period of time.**

I. Introduction

When we construct intelligent systems like person identification systems or stock price prediction systems, we often encounter a situation where a complete set of training samples is not given initially. In that case, we must construct a provisional system that works well in some limited conditions, and then the system will be improved incrementally using training samples given in the future. In this sense, incremental learning has been one of the most important issues in machine learning research so far.

The memory-based learning approach is one of the most promising strategy for incremental learning. In this approach, (almost) all training samples are accumulated in a memory, and then all of them are utilized for learning at every step. Locally Weighted Regression (LWR) [2] and Radial-Basis Function (RBF) networks [3], [4], [5] are successful examples of memory-based learning approach. In LWR, linear regression is conducted for all (or a part) of stored training samples so that a mean squared error weighted by the distance from a query point is minimized. Since LWR is substantially based on linear regression using only local samples around a query point, the training speed is quite fast. However, the regression might be inaccurate if a suitable weight function is not selected. On the other hand, RBF networks can learn any functions if the number of hidden units is sufficiently large, and the learning algorithm can be easily extended in an incremental fashion [4].

A serious drawback of the memory-based learning model of RBF networks is that it often needs large memory capacity to store the RBF centers. To avoid this problem, RBF centers should be selected automatically from training samples that are given incrementally. If sufficient training samples are initially given, several selection strategies like random selection and unsupervised selection of centers could be adopted [5]. In many incremental learning problems, however, these strategies are useless because we do not know when we will get sufficient training samples. The supervised selection of centers is another solution for RBF networks. In this approach, RBF centers are moved adaptively based on output errors and consequently the approximation accuracy could be held even if so many RBF centers are not allocated. In incremental settings, however, *catastrophic interference* might be a serious problem when the supervised selection of centers is introduced into RBF networks. This interference is caused by modifying connection weights and RBF centers, which hold the input-output relationships acquired from the past training samples, to adapt for new training samples.

To avoid the catastrophic interference, there have been proposed several approaches [6], [7], [8], [9]. A promising approach is that some representative input-output pairs are extracted from sequentially given training samples and some of them are trained with a current training sample. Based on this approach, we have proposed an extended version of RBF networks called *Resource Allocating Network with Long-Term Memory* (RAN-LTM) [9]. RAN-LTM does not need so much memory capacity and it realizes robust incremental learning ability. However, the training often needs long time because RBF centers are trained based on the gradient descent method.

In this paper, we propose a variant of RAN-LTM in which its centers are not trained but selected based on the output errors. Hence, the proposed RAN-LTM is a memory-based learning model of RBF networks in which the supervised selection of centers is introduced. In Section 2, the original RAN-LTM is briefly explained at first, and then the variant of RAN-LTM is presented in Section 3. To evaluate the learning speed and approximation accuracy, we apply it to some function approximation problems in Section 4. Finally we state conclusions and one of our future works in Section 5.

II. Resource Allocating Network with Long-Term Memory

A. Architecture of RAN-LTM

Figure 1 shows the architecture of RAN-LTM [9]. As seen from Fig. 1, RAN-LTM is composed of two modules: Resource Allocating Network (RAN) and Long-Term Memory

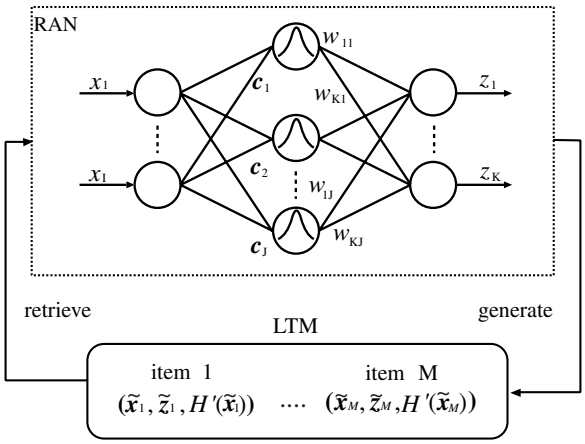

Fig. 1. Architecture of RAN-LTM.

(LTM). RAN learns desired input-output relations from given training samples. However, if training samples are given incrementally, RAN might forget the input-output relations acquired in the past due to 'catastrophic interference' caused by learning a new training sample. To avoid the forgetting, representative input-output pairs are extracted from the mapping function of RAN. Then, these pairs are stored in LTM and some of them are trained with a new training sample to suppress the interference. In the followings, these pairs stored in LTM are called 'memory items' for notational convenience.

B. Resource Allocating Network

Resource Allocating Network (RAN) proposed by Platt [10] is an extended version of RBF networks. RAN can adaptively add its hidden units to extend the approximation ability when unknown training samples are given.

Let the numbers of units in input, hidden, and output layers be I, J, and K, respectively. When an input $\boldsymbol{x} = \{x_1, \cdots, x_I\}'$ is given to RAN, the jth hidden output y_j and the kth network output are calculated as follows:

$$y_j = \exp\left(-\frac{\| \boldsymbol{x} - \boldsymbol{c}_j \|^2}{2\sigma_j^2}\right), \quad j = 1, \cdots, J \tag{1}$$

$$z_k = \sum_{j=1}^{J} w_{kj} y_j + \gamma_k, \quad k = 1, \cdots, K \tag{2}$$

where $\boldsymbol{c}_j = \{c_{j1}, \cdots, c_{jI}\}'$ and σ_j^2 are the center and variance of the jth hidden unit. And w_{kj} and γ_k are respectively a connection weight from the jth hidden unit to the kth output unit and a bias of the kth output unit.

In RAN, connection weights and RBF centers are modified to improve the approximation accuracy of network outputs. The modification of connection weights and centers is carried out based on the gradient descent method. After the error E between outputs $\boldsymbol{z}$ and targets $\boldsymbol{T}$ is evaluated, training of RAN is conducted as follows:

(1) If E is larger than a positive constant ε and the distance between input $\boldsymbol{x}$ and its nearest center vector $\boldsymbol{c}^*$ is larger than a positive value $\delta(t)$ (i.e., $E > \varepsilon$ and $\|\boldsymbol{x} - \boldsymbol{c}^*\| > \delta(t)$), then add a hidden unit to RAN (i.e., $J \leftarrow J + 1$). Set the following values to the network parameters for the Jth hidden unit (center vector $\boldsymbol{c}_J$, connection weights w_{kJ}, and variance σ_J):

$$c_{Ji} = x_i, \quad i = 1, \cdots, I \tag{3}$$

$$w_{kJ} = T_k - z_k, \quad k = 1, \cdots, K \tag{4}$$

$$\sigma_J = \kappa \|\boldsymbol{x} - \boldsymbol{c}^*\| \tag{5}$$

where κ is a positive constant. Decrease $\delta(t)$ with time t as follows:

$$\delta(t) = \max\left[\delta_{max} \exp\left(\frac{t}{\tau}\right), \delta_{min}\right] > 0 \tag{6}$$

where τ is a decay constant.

(2) Otherwise

Modify the network parameters as follows:

$$w_{kj}^{NEW} = w_{kj}^{OLD} + \alpha(T_k - z_k) y_j \tag{7}$$

$$c_{ji}^{NEW} = c_{ji}^{OLD} + \frac{\alpha}{\sigma_j}(x_i - c_{ji}) y_j \sum_k (T_k - z_k)\sigma_{kj} \tag{8}$$

$$\gamma_k^{NEW} = \gamma_k^{OLD} + \alpha(T_k - z_k) \tag{9}$$

where α is a positive learning ratio.

C. Long-Term Memory

Another module in RAN-LTM is Long-Term Memory (LTM). Representative input-output pairs are extracted from the mapping function acquired in RAN and they are stored in LTM over a long period of time. These pairs are called 'memory items' and some of them are retrieved from LTM to learn with an immediate training sample. The learning of memory items with a training sample prevents RAN from forgetting the mapping function acquired in the past even if training samples are given incrementally.

Although all memory items could be always retrieved to learn, it is generally desirable that the number of retrieved memory items are restricted only to effective ones. To estimate the effectiveness of memory items, we adopt the following two criteria: (1) the activation of hidden units and (2) the curvature information of the approximated function. Since the weights and RBF centers are modified in proportion to the activation of hidden units, it is considered that the input-output relations in the neighborhood of RBF centers for the active hidden units tend to be deteriorated by the interference. The first criterion is based on this fact. The second criterion is based on our previous experimental results [9]: memory items on high curvature parts of approximated functions are more useful for suppressing the interference as compared with those on low curvature points. Considering these facts, we can select only the nearest memory items to the centers for active hidden units and on high curvature parts of approximated functions.

Hence, we shall retrieve memory items based on the following probability:

$$P_j = \frac{1}{1+\exp[-\nu\{y_j + H'(\tilde{\boldsymbol{x}}_j)\}+\lambda]} \tag{10}$$

where ν and λ are positive constants, and y_j is the output of the jth hidden unit. $H'(\tilde{\boldsymbol{x}}_j)$ is the following Hessian information with respect to the nearest memory item $\tilde{\boldsymbol{x}}_j$ to the jth RBF center:

$$H'(\tilde{\boldsymbol{x}}_j) = \min\{\frac{\sum_k |H_k(\tilde{\boldsymbol{x}}_j)|}{H_0}, 1\} \tag{11}$$

where H_0 is a positive constant and $H_k(\cdot)$ is the determinant of the Hessian matrix of the output $\tilde{z}_k$ whose element (i, i') is given as follows:

$$\frac{\partial^2 \tilde{z}_k}{\partial c_{ji}\partial c_{ji'}} = \frac{1}{\sigma_i^2\sigma_{i'}^2}\sum_{l=1}^{J} w_{kl}(c_{ji}-c_{li})(c_{ji'}-c_{li'})y_l + \gamma_k. \tag{12}$$

The memory items $\tilde{\boldsymbol{x}}_j$ are often sparsely distributed in a problem space. In such a case, learning these memory items could cause new interference even though they are trained to suppress the interference for a training sample. A remedy for this new interference is that some neighbor points of retrieved memory items are also trained to hold the input-output relations in the neighborhood of the memory items. These neighborhood points can be obtained as $(\tilde{\boldsymbol{x}}_j + \Delta, g(\tilde{\boldsymbol{x}}_j + \Delta))$ where Δ is a small random value and $g(\cdot)$ is the mapping function of RAN.

The detailed procedures of generating and retrieving memory items are shown below.

[Procedure of Generation]

(1) When a training sample $(\boldsymbol{x}, \boldsymbol{T})$ is given to RAN, calculate the hidden outputs, y_j $(j = 1, \cdots, J)$.
(2) If all hidden outputs are less than a threshold value θ_c, then go to Step 6. Otherwise, go to Step 3.
(3) Obtain all indices j of hidden units whose outputs y_j are larger than θ_c, and define a set $\mathcal{I}_1$ of these indices. Update the following approximation criterion r_j for all $j \in \mathcal{I}_1$:

$$r_j^{NEW} = r_j^{OLD} + 2\exp(-\rho|E|) - 1$$

where E is the output error and ρ is a positive constant. If $r_j > \beta$ for $j \in \mathcal{I}_1$, go to Step 4. Otherwise, go to Step 6. Here, β is a positive constant.
(4) Calculate the minimum distance between the jth center $\boldsymbol{c}_j$ and memory items $\tilde{\boldsymbol{x}}_m$ $(m = 1, \cdots, M)$ as follows:

$$d_j^* = \min_m \|\boldsymbol{c}_j - \tilde{\boldsymbol{x}}_m\|.$$

If $d_j^* > \eta$ for $j \in \mathcal{I}_1$, then go to Step 5. Otherwise go to Step 6.
(5) Increase the number of memory items M by one (i.e., $M \leftarrow M + 1$). Give the jth center vector $\boldsymbol{c}_j$ to RAN as its input, and obtain the output $\boldsymbol{z}$. Calculate the Hessian information $H'(\boldsymbol{c}_j)$ in Eq. (11). Store the triplet $(\boldsymbol{c}_j, \boldsymbol{z}, H'(\boldsymbol{c}_j))$ in LTM as the Mth memory item $(\tilde{\boldsymbol{x}}_M, \tilde{\boldsymbol{z}}_M, H'(\tilde{\boldsymbol{x}}_M))$.
(6) Go back to Step 1.

[Procedure of Retrieval]

(1) When a training sample $\boldsymbol{x}$ is given to RAN, calculate hidden outputs, y_j $(j = 1, \cdots, J)$.
(2) For the jth hidden unit, find a memory item $(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{z}}_j, H'(\tilde{\boldsymbol{x}}_j))$ whose input vector $\tilde{\boldsymbol{x}}_j$ is the nearest to the jth RBF centers $\boldsymbol{c}_j$. Repeat the above operation for all hidden units.
(3) For each hidden unit, calculate the recall probability P_j from y_j and $H'(\tilde{\boldsymbol{x}}_j)$ from Eq. (10).
(4) Generate pseudo training samples, $(\hat{\boldsymbol{x}}_l, \hat{\boldsymbol{z}}_l)$ $(l = 1, \cdots, L_j)$, as follows:

$$\begin{aligned} \hat{\boldsymbol{x}}_l &= \tilde{\boldsymbol{x}}_j + \Delta_l \\ \hat{\boldsymbol{z}}_l &= g(\tilde{\boldsymbol{x}}_l) \end{aligned}$$

where L_j is the number of pseudo training samples for the jth memory item.
(5) Retrieve input-output pairs of memory items, $(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{z}}_j)$ $(j = 1, \cdots, J)$, with probability P_j, and pseudo training samples, $(\hat{\boldsymbol{x}}_l, \hat{\boldsymbol{z}}_l)$ $(l = 1, \cdots, L_j)$.
(6) Go back to Step 1.

III. A Fast Incremental Learning Algorithm of RAN-LTM

A. Learning Strategy of RBF Networks

As described in Section I, many learning strategies for RBF networks have been proposed so far [5]. They can be categorized into two groups in terms of whether RBF centers are trained or not.

If RBF centers are not trained, the selection strategy of centers is important. Unfortunately, however, many selection strategies can be used only for batch learning. Therefore, if we adopt RBF networks with fixed centers in incremental learning problems, (almost) all of the incrementally given training samples should be allocated to networks as their centers. This is often called memory-based learning, and it would need much time to learn and large memory capacity to save RBF centers when the number of training samples is huge. If we can select essential training samples in incremental learning settings, the learning could be very fast because the training of hidden-output connection weights is reduced to solving a set of simultaneous linear equations [4], [5].

On the other hand, RBF centers in RAN-LTM are trained based on the gradient descent algorithm. Hence, the training speed is not so fast in general; however, RAN-LTM generally attains desired mapping functions with a small number of RBF centers. There are two main reasons for this: one reason is due to the adaptability of centers and the other reason is that centers are automatically allocated in only poorly approximated regions based on output errors. In this context, we come up with a variant type of RAN-LTM, in which RBF centers are fixed and the selection is done by the automatic

allocation mechanism of hidden units in RAN. This will realize both speedup of learning and small memory capacity in RAN-LTM.

B. Fast Learning Algorithm of RAN-LTM

A simple implementation of the above RAN-LTM is that RBF centers are stored in LTM, and the recall probabilities P_j in Eq. (10) for all memory items are set to 1. The proposed learning algorithm based on this simple implementation is shown as follows:

[Learning Algorithm]

(1) Find the nearest center $\boldsymbol{c}^*$ to an input $\boldsymbol{x}$ and then calculate the output error E.

(2) If $E > \varepsilon$ and $\|\boldsymbol{x} - \boldsymbol{c}^*\| > \delta(t)$, then add a hidden unit and generate a memory item $(\tilde{\boldsymbol{x}}_M, \tilde{\boldsymbol{z}}_M)$ as follows:

$$\boldsymbol{w}_J = \boldsymbol{T} - \boldsymbol{z}, \qquad \boldsymbol{c}_J = \boldsymbol{x}$$

$$\tilde{\boldsymbol{x}}_M = \boldsymbol{x}, \qquad \tilde{\boldsymbol{z}}_M = \boldsymbol{T}.$$

Go to Step 8. Otherwise, go to Step 3.

(3) Recall all memory items, $(\tilde{\boldsymbol{x}}_m, \tilde{\boldsymbol{z}}_m)$ $(m = 1, \cdots, M)$, from LTM.

(4) For each memory item, generate pseudo training samples, $(\hat{\boldsymbol{x}}_l, \hat{\boldsymbol{z}}_l)$ $(l = 1, \cdots, L_m)$, as follows:

$$\begin{aligned} \hat{\boldsymbol{x}}_l &= \tilde{\boldsymbol{x}}_m + \Delta_l \\ \hat{\boldsymbol{z}}_l &= g(\tilde{\boldsymbol{x}}_l) \end{aligned}$$

where L_m is the number of pseudo training samples for the mth memory item, $g(\cdot)$ means the mapping function of RAN, and Δ_l is a randomly selected small value.

(5) Calculate hidden outputs for all memory items, $\phi(\tilde{\boldsymbol{x}}_m) = \{\phi(\tilde{x}_{m1}), \cdots, \phi(\tilde{x}_{mJ})\}$, and hidden outputs for pseudo samples, $\phi(\hat{\boldsymbol{x}}_l) = \{\phi(\hat{x}_{l1}), \cdots, \phi(\hat{x}_{lJ})\}$ $(l = 1, \cdots, L_m)$. Obtain an $N \times J$ matrix $\boldsymbol{\Phi}$ whose row vectors correspond to $\phi(\tilde{\boldsymbol{x}}_m)$ and $\phi(\hat{\boldsymbol{x}}_l)$. Here, N is the total number of memory items and pseudo samples.

(6) Using the singular value decomposition (SVD), decompose $\boldsymbol{\Phi}$ as follows:

$$\boldsymbol{\Phi} = \boldsymbol{U}\boldsymbol{H}\boldsymbol{V}'$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $N \times J$ and $J \times J$ orthogonal matrices, and $\boldsymbol{H}$ is a $J \times J$ diagonal matrix. Then calculate a weight matrix $\boldsymbol{W}$ as follows:

$$\boldsymbol{W} = \boldsymbol{V}\boldsymbol{H}^{-1}\boldsymbol{U}'\boldsymbol{D}.$$

(7) Give the input $\boldsymbol{x}$ to RAN-LTM again, and calculate the output error E. If $E > \varepsilon$, then add a hidden unit and generate a memory item $(\tilde{\boldsymbol{x}}_M, \tilde{\boldsymbol{z}}_M)$ as follows:

$$\boldsymbol{w}_J = \boldsymbol{T} - \boldsymbol{z}, \qquad \boldsymbol{c}_J = \boldsymbol{x}$$

$$\tilde{\boldsymbol{x}}_M = \boldsymbol{x}, \qquad \tilde{\boldsymbol{z}}_M = \boldsymbol{T}.$$

(8) The learning is over. When a new training sample is given, go back to 1.

IV. Simulations

To evaluate the proposed learning algorithm of RAN-LTM, it is applied to function approximation problems under the condition that training samples are incrementally given. For comparison purposes, the following four models are evaluated:

i) RBFN
The conventional RBF network in which all training samples are set to its RBF centers. The weight connections are obtained by solving a set of simultaneous linear equations. When a hidden unit is added to the network, the incremental operation for the matrix inversion is also introduced here (see [4] for details).

ii) RAN-LTM (O)
The original RAN-LTM in which training is carried out based on the gradient descent method (see Section II).

iii) RAN-LTM (N1)
The proposed RAN-LTM in which training is carried out based on the fast learning algorithm (see Subsection III-B), but pseudo training samples are not utilized for learning.

iv) RAN-LTM (N2)
The proposed RAN-LTM in which learning is carried out based on the fast learning algorithm (see Subsection III-B). Here, we randomly generate three pseudo training samples per RBF center.

RAN-LTM (N1) is adopted here to examine the effectiveness of learning pseudo training samples as well as memory items. The training time and approximation accuracy are investigated in the following two problems: one-dimensional function approximation and prediction of Mackey-Glass time series.

A. Experiment 1

The target function to be approximated is the following one-dimensional function (see also Fig. 2):

$$z = f(x) = \frac{20\sin(4\pi(x + 0.5)\cos(10\pi(x + 0.5)))}{x + 10}. \quad (13)$$

Training samples (x, z) are randomly drawn from the function in Eq. (13), and they are incrementally given to a network. To examine the performance depending on the size of problems, three different regions are defined for input x: R1: $\{x \mid 0 < x < 20\}$, R2: $\{x \mid 0 < x < 40\}$ and R3: $\{x \mid 0 < x < 80\}$. The numbers of training samples in R1, R2 and R3 are 500, 1000, and 2000, respectively. The approximation accuracy is estimated for test samples after incremental learning is completed. The test samples are also randomly drawn from the same regions, and the numbers of them are 1000, 2000, and 4000, respectively.

Tables I (a)-(c) show the mean squared errors for training and test samples, and also show the CPU time for convergence and the maximum memory size needed for learning of R1, R2, and R3. To estimate the memory size, we consider almost all factors to be stored in the main memory such as connection

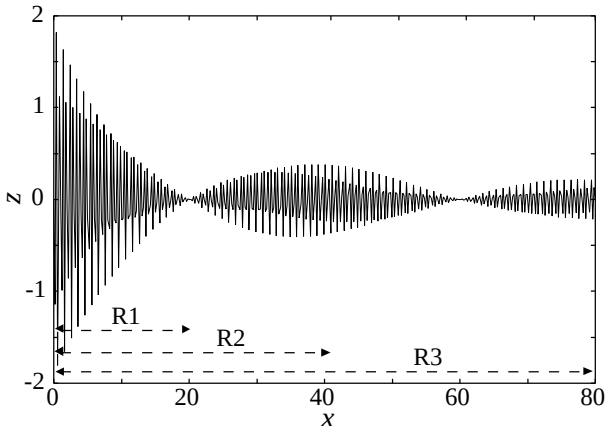

Fig. 2. One-dimensional function $f(x)$.

TABLE I

RESULTS FOR ONE-DIMENSIONAL APPROXIMATION FUNCTION.

(a) Training of R1 (500 training samples)

MODEL	Training Error	Test Error	Time (min.)	Memory (KB)
RBFN	0	0.03976	18.5	2012
RAN-LTM (O)	0.54056	0.54111	30.4	6
RAN-LTM (N1)	0.40817	0.41016	1.5	133
RAN-LTM (N2)	0.15264	0.15886	15.1	823

(b) Training of R2 (1000 training samples)

MODEL	Training Error	Test Error	Time (min.)	Memory (KB)
RBFN	0	0.02285	284	8024
RAN-LTM (O)	0.42472	0.42497	122	11
RAN-LTM (N1)	0.24674	0.24829	32.4	442
RAN-LTM (N2)	0.14118	0.14538	157	1897

(c) Training of R3 (2000 training samples)

MODEL	Training Error	Test Error	Time (min.)	Memory (KB)
RBFN	0	0.01241	4247	32048
RAN-LTM (O)	0.20646	0.20653	431	21
RAN-LTM (N1)	0.17047	0.17092	486	1465
RAN-LTM (N2)	0.09979	0.10192	1694	5743

weights, centers and variances of hidden units, interpolation matrices for SVD calculations, and memory items.

As seen from Tables I (a)-(c), RBFN realizes very high approximation accuracy for both training and test samples. This is because all training samples are stored as RBF centers and the output errors are always minimized using all samples. However, training time increases exponentially as the number of training samples becomes large.

The accuracy of RAN-LTM (O) is not good especially when the number of training samples is small. Considering that the learning algorithm is based on the gradient descent method, the deterioration in accuracy might be due to the convergence on a local minimum. On the other hand, the approximation accuracy of RAN-LTM (N1) and RAN-LTM (N2) is rather good. Hence, one can say that the supervised selection of RBF centers contributes to realizing good generalization ability in RAN-LTM.

The learning of RAN-LTM (N1) is very efficient unless the number of training samples is too large. As seen from Tables I(a)-(c), however, the learning time of RAN-LTM (N1) and RAN-LTM (N2) increases distinctively when the number of training samples is larger than 1000. Interestingly, when the number of training samples is 2000, RAN-LTM (O) learns faster than both RAN-LTM (N1) and RAN-LTM (N2) even if the learning algorithm is based on the gradient descent method. The computational complexity of SVD calculations in these two RAN-LTMs is approximately $O(J^3)$ where J is the number of hidden units. Therefore, this result shows that the computational costs for SVD calculations could be expensive for large incremental learning problems even if only essential RBF centers are selected.

As seen from Tables I(a)-(c), RBFN needs large main memory to carry out learning as compared with the other models even in the one-dimensional function approximation problems. Hence, we can say that if the accuracy is considered as a precedence factor and there are enough memory resources, RBFN should be adopted here. However, if the memory resources are limited, the other three RAN-LTM models are worth adopting especially when a large number of training samples are incrementally given.

B. Experiment 2

Here, we apply the variant of RAN-LTM to a multi-dimensional function approximation problem: Mackey-Glass time series forecasting. Mackey-Glass time series is given by the following differential equation:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t). \tag{14}$$

From Eq. (14), we can obtain the time series data $x(0)$, $x(1)$, $x(2)$, $\cdots$, $x(t)$, $\cdots$. Using x prior to time t, we predict x after t. In this experiment, setting $\tau = 17$, and using four inputs $x(t-18)$, $x(t-12)$, $x(t-6)$, $x(t)$, we estimate $x(t+6)$. In this way we generate a thousand data $x(118), \cdots, x(1117)$. These time series data are depicted in Fig. 3. The four models are applied to the function approximation problem using such Mackey-Glass time series data.

The first 500 data points are used as training samples, and the remaining 500 data points are used to test performance. The training sample is incrementally given to neural networks at random. Table II shows the experimental results in this simulation.

As you can see from Table II, the training of RAN-LTM (N1) and RAN-LTM (N2) is very fast and the errors for both training and test samples are smaller than the conventional RBFN. RAN-LTM (O) also has good approximation accuracy, but it takes quite long time for learning as compared with RAN-LTM (N1) and RAN-LTM (N2). For test samples, the prediction performance of RBFN is very poor due to the over learning of training samples. In this experiment, the number of hidden units in RBFN is 500, while the numbers of hidden

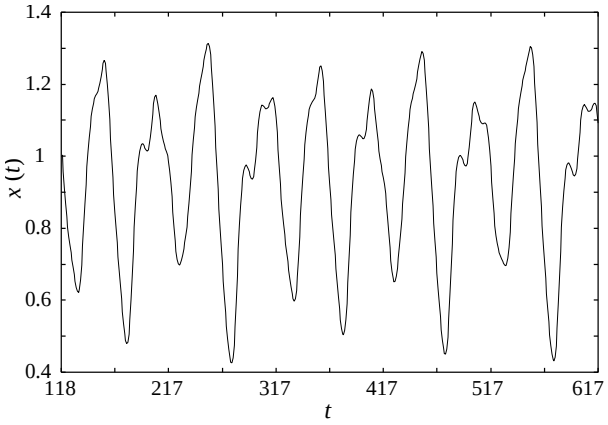

Fig. 3. Mackey-Glass time series.

TABLE II
EXPERIMENTAL RESULTS FOR MACKEY-GLASS TIME SERIES.

MODEL	Training Error	Test Error	Time (sec.)	Memory (KB)
RBFN	0.00307	0.75086	1012.6	2024
RAN-LTM (O)	0.00316	0.00316	261.4	1
RAN-LTM (N1)	0.00167	0.00162	0.22	1.5
RAN-LTM (N2)	0.00125	0.00120	0.48	1.9

units in the three RAN-LTM models are less than 10. This result suggests that the number of hidden units should be selected to obtain good generalization. In many incremental learning settings, however, the conventional RBFN can not select suitable number of hidden units. In this sense, we can say that the supervised selection of centers is a good choice in incremental learning problems. Furthermore, RBFN needs a large memory capacity to store the information of RBF centers. It seems that the training of pseudo samples in RAN-LTM (N2) is also effective if we need high accuracy in approximation. Instead, it takes longer time in training.

V. CONCLUSIONS

In this paper, we proposed a fast incremental learning algorithm of Resource Allocating Network with Long-Term Memory (RAN-LTM), in which its centers are not trained but selected based on output errors (i.e., supervised selection of RBF centers). This model does not need so much memory capacity and it realizes robust incremental learning ability. To verify these characteristics of the modified RAN-LTM, we applied it to two function approximation problems: one-dimensional function approximation and prediction of Mackey-Glass time series. In both experiments, the training samples were given to the networks incrementally.

From the experimental results, we verified that the proposed RAN-LTM can learn fast and accurately without large main memory unless the number of training samples becomes too large. However, the computational costs for SVD calculations could be a problem in RBFN and the proposed RAN-LTM if incremental learning continues for long time. In this case, the original RAN-LTM, in which RBF centers are trained, could be rather effective. Furthermore, the training of pseudo samples as well as memory items in RAN-LTM (N2) is also effective if we need high accuracy in approximation, although it might take longer time in learning.

To simplify the learning algorithm as much as possible, we implemented a fast incremental learning of RAN-LTM in which all memory items are always retrieved to learn with a training sample. However, all memory items might not be always needed for recalling. Introducing an efficient retrieval mechanism into the proposed variant of RAN-LTM will lead to a faster incremental learning algorithm. This is left as our future works.

ACKNOWLEDGMENT

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B).

REFERENCES

[1] G.A. Carpenter and S. Grossberg: "The ART of adaptive pattern recognition by a self-organizing neural network," *IEEE Computer*, **21**, 3, 77/88 (1988)
[2] C. G. Atkeson, A. W. Moore and S. Schaal: "Locally weighted learning," *Artificial Intelligence Review*, **11**, 75/113 (1997)
[3] T. Poggio and F. Girosi: "Networks for approximation and learning," *IEEE Trans. on Neural Networks*, **78**, 9, 1481/1497 (1990)
[4] M. J. L. Orr: "Introduction to radial basis function networks," *Technical Report of Institute for Adaptive and Neural Computation*, Division of Informatics, Edinburgh University (1996)
[5] S. Haykin: *Neural Networks-A Comprehensive Foundation* (2nd Ed.), Prentice Hall (1999)
[6] H. Nakayama and M. Yoshida: "Additional learning and forgetting by potential method for pattern classification," *Proc. Int. Conf. on Neural Networks 97*, 1839/1844 (1997)
[7] K. Yamauchi, N. Yamaguchi, and N. Ishii: "Incremental learning methods with retrieving of interfered patterns," *IEEE Trans. on Neural Networks*, **10**, 6, 1351/1365 (1999)
[8] H.-C. Fu, Y.-P. Lee, C.-C. Chiang, and H.-T. Pao: "Divide-and-conquer learning and modular perceptron networks," *IEEE Trans. on Neural Networks*, **12**, 2, 250/263 (2001)
[9] M. Kobayashi, A. Zamani, S. Ozawa and S. Abe: "Reducing computations in incremental learning for feedforward neural network with long-term memory," *Proc. Int. Joint Conf. on Neural Networks*, 1989/1994 (2001)
[10] J. Platt: "A resource allocating network for function interpolation," *Neural Computation*, **3**, 213/225 (1991)