Function Approximation Based on a Network with Kernel Functions of Bounds and Locality : an Approach of Non-Parametric Estimation

by Rhee M. Kil

This paper presents function approximation based on nonparametric estimation. As an estimation model of function approximation,a three layered network composed of input, hidden and output layers is considered. The input and output layers have linear activation units while the hidden layer has nonlinear activation units or kernel functions which have the characteristics of bounds and locality. Using this type of network, a many-to-one function is synthesized over the domain of the input space by a number of kernel functions. In this network, we have to estimate the necessary number of kernel functions as well as the parameters associated with kernel functions. For this purpose, a new method of parameter estimation in which linear learning rule is applied between hidden and output layers while nonlinear (piecewise-linear) learning rule is applied between input and hidden layers, is considered.The linear learning rule updates the output weights between hidden and output layers based on the Linear Minimization of Mean Square Error (LMMSE) sense in the space of kernel functions while the nonlinear learning rule updates the parameters of kernel functions based on the gradient of the actual output of network with respect to the parameters (especially, the shape) of kernel functions.This approach of parameter adaptation provides near optimal values of the parameters associated with kernel functions in the sense of minimizing mean square error.

As a result, the suggested nonparametric estimation provides an efficient way of function approximation from the view point of the number of kernel functions as well as learning speed.

I. Introduction

An artificial neural network can be evaluated in terms of its capability of accurately representing a desired input-output mapping through efficient training of a given set of teaching patterns. An accurate representation of a mapping depends on the proper selection of a network configuration including the network architecture, the number of

neurons and the type of activation functions, and the capability of a learning algorithm to find the optimal parameters for the selected network configuration.

Most artificial neural networks developed up to date have focused on training the parameters of a fixed network configuration selected by the designer. However, it may be an extremely powerful tool for constructing an optimal network, if a learning algorithm has a capability of automatically configuring a neural network, in addition to the adjustment of network parameters. Although attempts have been made to apply the idea of self-recruiting neurons to the automatic clustering of input samples [1] and to the identification of class boundaries [2], a major effort needs to be expended to establish a learning algorithm capable of automatically configuring a network based on the self-recruitment of neurons with a proper type of activation functions. As an effort of such approach, a non-sigmoidal *Mapping Neural Network* (MNN) [3], called the "*Potential Function Network* (PFN)" [4, 5, 6] was presented. The PFN is capable of approximating a "many-to-one"continuous function by a potential field synthesized over the domain of the input space by a number of computational units called "*Potential Function Units* (PFUs)". Recently, Niranjan and Fallside [7], and Moody and Darken [8] successfully train the continuous functions using a three-layer network with hidden units which have localized receptive fields (or Radial Basis Functions). However, their approach is lacking the flexibility of determining the proper number of hidden units according to the desired level of accuracy in the function approximation. In PFN, the emphasis is given to the synthesis of a potential field based on a new type of learning called the "*Hierarchically Self-Organizing Learning* (HSOL)" [6]. The distinctive feature of HSOL is its capability of automatically recruiting necessary PFUs under the paradigm of hierarchical learning, implemented through the successive adjustment of the accommodation boundaries or the effective radii of individual PFUs in the input domain.

The parameter adaptation in HSOL was based on *Error-Back-Propagation* (EBP) algorithm [9]. However, EBP algorithm does not guarantee convergence and generally suffers slow learning. In this point of view, a new method of parameter estimation in which linear learning rule is applied between hidden and output layers while nonlinear (piecewise-linear) learning rule is applied between input and hidden layers, is considered. The linear learning rule updates the output weights between hidden and output layers based on the Linear Minimization of Mean Square Error (LMMSE) sense in the space of kernel functions while the nonlinear learning rule updates the parameters of kernel functions based on the gradient of the actual output of network with respect to the parameters (especially, the shape) of kernel functions. This approach of parameter adaptation provides near optimal values of the parameters associated with kernel functions in the sense of minimizing mean square error. As a result, the suggested nonparametric estimation provides an efficient way of function approximation from the view point of the number of kernel functions as well as learning speed.

This paper is organized as follows: in section I, an estimation model, a network with kernel functions of bounds and locality, is suggested and proved to be an universal function approximator; in section III, a new learning algorithm comprising the automatic recruitment of kernel functions as well as the parameter estimation of a network, is suggested; in section IV, the simulation of the suggested learning algorithm is shown for various cases of test samples; and finally, the section V

addresses the conclusion.

II. Estimation Model

It has been proposed that a discriminant function, $\phi(\mathbf{x})$, can be represented by the weighted summation of a finite number of *potential functions* [4] as follows:

$$\phi(\mathbf{x}) = \sum_{i=1}^{M} c_i K(\mathbf{x}, \mathbf{x}_i) \qquad (1)$$

where $K(\mathbf{x}, \mathbf{x}_i)$ is the ith potential function of $\mathbf{x}$, obtained by shifting $K(\mathbf{x}, \mathbf{0})$ by $\mathbf{x}_i$, and c_i is a real constant. For instance, the potential function $K(\mathbf{x}, \mathbf{x}_i)$ of classical physics varies inversely with $\| \mathbf{x} - \mathbf{x}_i \|$, i.e., $K(\mathbf{x}, \mathbf{x}_i)$ has the maximum value at $\mathbf{x} = \mathbf{x}_i$ and decreases monotonically to zero as $\| \mathbf{x} - \mathbf{x}_i \|$ approaches infinity.

Here, the potential function is selected based on the following condition:

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{\infty} \varphi_i(\mathbf{x}) \varphi_i(\mathbf{y}) \text{ and } \max_{\mathbf{x}} K(\mathbf{x}, \mathbf{x}) \le L \qquad (2)$$

where the function system $\{\varphi_i(\mathbf{x})\}$ is orthonormal in the space of $\mathbf{x}$ and L is a bounded scalar.

To train the given function from the teaching samples, a learning algorithm similar to that of the *Perceptron* [10] has been proposed for applying (1) to binary classification:

$$\phi^{new}(\mathbf{x}) = \begin{cases} \phi^{old}(\mathbf{x}) + K(\mathbf{x}, \mathbf{x}_k) & \text{if the sample, } \mathbf{x}_k \text{ is labelled +1 and } \phi^{old}(\mathbf{x}_k) \le 0 \\ \phi^{old}(\mathbf{x}) - K(\mathbf{x}, \mathbf{x}_k) & \text{if the sample, } \mathbf{x}_k \text{ is labelled -1 and } \phi^{old}(\mathbf{x}_k) \ge 0 \\ \phi^{old}(\mathbf{x}) & \text{otherwise} \end{cases} \qquad (3)$$

It has been shown that (3) converges within finite steps.

The potential function approach to binary classification described by(1) and (3) has a similar flavor to the nonparametric estimation of a probability density function based on the *Parzen window* [11]. In the Parzen window approach, a probability density function, $p(\mathbf{x})$, is estimated from the observed input samples, $\mathbf{x}_i$s, $i = 1, \cdots, n$, by

$$p_n(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h_n^d} \psi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_n}\right) \qquad (4)$$

where ψ represents a bounded nonnegative kernel function of the d dimensional input vector, $\mathbf{x}$, and h_n is a sequence of positive numbers such that $\lim_{n\to\infty} h_n = 0$ and $\lim_{n\to\infty} n h_n^d = \infty$. It can be shown from (4) that $p_n(\mathbf{x})$ converges to $p(\mathbf{x})$ as n approaches ∞.

The problem associated with (1) or (4) is that the number of potential functions or kernel functions required for implementing an unknown function becomes potentially very large proportional to the number of input samples. This is due to the fact that (1) or (4) is based on the shifted summation of prespecified shape (variance) of the potential or kernel functions assigned to individual input samples. This problem may be resolved by relaxing the fundamental constraints associated with (1) or (4): the position shift of a kernel function should correspond to the coordinate of input samples and the shape of a kernel function should be fixed, and by introducing a methodology of self-recruiting a minimum necessary number of kernel functions with the capability of adjusting both the position shift and the shape parameters of individual kernel functions.

A generalized form of (1) or (4), incorporating

the adjustment of shape parameters and the self-recruitment of kernel functions, can be expressed as

$$\phi(\mathbf{x}) = \sum_{i=1}^{M} c_i \psi(\mathbf{x}, \mathbf{p}_i) \qquad (5)$$

where M represents the number of kernel functions to be recruited, c_i represents the summation weight, and $\mathbf{p}_i$ represents a new parameter vector including both the position shift and the shape parameters of the ith kernel function. In (5), M, c_i and $\mathbf{p}_i$, i=1, ···, M, are subject to the adjustment through learning. (5) may be able to achieve a desirable error level in function approximation with a smaller number of kernel functions, but may require a more complicated learning algorithm.

According to Funahashi [12] and Hornik, Stinchcombe and White [13], (5) can approximate a continuous function with a desirable degree of accuracy based on a sufficiently large number of hidden units, provided ψ is an absolutely integrable or a bounded monotonic (squashing) function. Here, let us investigate the mapping capability of a network using kernel functions which have the following properties:

1. Bounds:

$$L_\psi \le \psi(\mathbf{x}, \mathbf{p}) \le U_\psi \qquad (6)$$

where L_ψ and U_ψ represent the lower and upper bounds of $\psi(\mathbf{x}, \mathbf{p})$ respectively.

2. Locality:

$$\int_{-\infty}^{+\infty} (\frac{1}{a})^N \psi(\mathbf{x}, \mathbf{p}_a) d\mathbf{x} = C \qquad (7)$$

$$\lim_{a \to \infty} (\frac{1}{a})^N \psi(\mathbf{x}, \mathbf{p}_a) = C\delta(\mathbf{x}) \qquad (8)$$

where N is the dimension of $\mathbf{x}$ and C is a positive constant.

There are many functions satisfying the above conditions. The examples of such kernel functions are

$$f(x,a) = e^{-\frac{\pi x^2}{a^2}} \qquad (9)$$

$$f(x,a) = Sinc(\frac{x}{a}) = \frac{a}{\pi x} sin(\frac{\pi x}{a}) \qquad (10)$$

$$f(x,a) = \mathrm{Re}ct(\frac{x}{a}) = \begin{cases} 1 & \text{if } |x| \le \frac{a}{2} \\ 0 & \text{otherwise} \end{cases} \qquad (11)$$

etc. Note that the above functions satisfy $\lim_{a \to 0} \frac{1}{a} f(x,a) = \delta(x)$. Note also that the *Radial Basis Function*, g ; $\mathbf{R}^n \to \mathbf{R}$, $g(\mathbf{x}) = H(\| \mathbf{x} - \mathbf{x}_\alpha \|)$, where H is some smooth real function of distance, $\| \mathbf{x} - \mathbf{x}_\alpha \|$ from the center $\mathbf{x}_\alpha$ in the input space, can be used as a candidate of kernel functions. Here, we suggest the following theorem claiming the network using kernel functions of bounds and locality as the universal approximator.

Theorem 1 *Every continuous function $f(\mathbf{x})$ defined on a compact set $\mathbf{R}^N$, can be approximated by a network represented by (5), i.e., for $\varepsilon > 0$, there exists a $\phi(\mathbf{x})$ such that*

$$|f(\mathbf{x}) - \phi(\mathbf{x})| \le \varepsilon \quad \forall \mathbf{x} \in \mathbf{R}^N \qquad (12)$$

Proof Let us define $f^*(\mathbf{x})$ as

$$f^*(\mathbf{x}) = f(\mathbf{x}) * (\frac{1}{a})^N \psi(\mathbf{x}, \mathbf{p}_a)$$
$$= \int_{-\infty}^{+\infty} f(\xi) (\frac{1}{a})^N \psi(\mathbf{x} - \xi, \mathbf{p}_a) d\xi \qquad (13)$$

and the integration of kernel function multi-

plied by $(\frac{1}{a})^N$ is normalized to 1 for the convenience, i.e.,

$$\int_{-\infty}^{+\infty}(\frac{1}{a})^N \psi(\mathbf{x},\mathbf{p}_a)d\mathbf{x} = 1. \qquad (14)$$

By choosing a small, $f^*(\mathbf{x})$ can be approximated by

$$f^*(\mathbf{x}) \sim \int_{|\mathbf{x}-\xi|\le a} f(\xi)(\frac{1}{a})^N \psi(\mathbf{x}-\xi,\mathbf{p}_a)d\xi. \qquad (15)$$

Thus, the difference between $f(\mathbf{x})$ and $f^*(\mathbf{x})$ is given by

$$f(\mathbf{x}) - f^*(\mathbf{x}) \sim \int_{|\mathbf{x}-\xi|\le a}(f(\mathbf{x}) - f(\xi))(\frac{1}{a})^N \psi(\mathbf{x}-\xi,\mathbf{p}_a)d\xi. \qquad (16)$$

Let us define $\varepsilon_1(\mathbf{x}, a)$ as

$$\varepsilon_1(\mathbf{x}, a) = |f(\mathbf{x}) - f(\xi)| \text{ for } \mathbf{x} \in I_1 = \{\, \mathbf{x} \,|\, |\mathbf{x} - \xi| \le a \,\}. \qquad (17)$$

By the assumption of continuity of $f(\mathbf{x})$, we can safely assume that

$$\varepsilon_1(\mathbf{x}, a) \le \in_1(a) \qquad (18)$$

where $\in_1(a)$ is a small positive constant. Note that $\lim_{a\to 0} \in_1(a) = 0$.

This implies,

$$|f(\mathbf{x}) - f^*(\mathbf{x})| \le \in_1(a) \int_{|\mathbf{x}-\xi|\le a}(\frac{1}{a})^N \psi(\mathbf{x}-\xi,\mathbf{p}_a)d\xi \le \in_1(a). \qquad (19)$$

Now, let us approximate $f^*(\mathbf{x})$ as the Riemann integral form, i.e.,

$$\begin{aligned} f^*(\mathbf{x}) &= \int_{-\infty}^{+\infty} f(\xi)(\frac{1}{a})^N \psi(\mathbf{x}-\xi,\mathbf{p}_a)d\xi \\ &= \sum_{\mathbf{x}_k \in I_2} f(\mathbf{x}_k)(\frac{1}{a})^N \psi(\mathbf{x}-\mathbf{x}_k,\mathbf{p}_a)a^N + \varepsilon_2(\mathbf{x},a) \\ &= \sum_{\mathbf{x}_k \in I_2} f(\mathbf{x}_k)\psi(\mathbf{x}-\mathbf{x}_k,\mathbf{p}_a) + \varepsilon_2(\mathbf{x},a) \end{aligned} \qquad (20)$$

where $\mathbf{x}_k$ is the point of a square grid of spacing a, I_2 is the finite set of lattice points and $\varepsilon_2(\mathbf{x},a)$ is the discretization error with the property of $\lim_{a\to 0} \varepsilon_2(\mathbf{x}, a) = 0$.

Let us define $\in_2(a)$ as

$$\in_2(a) = \max_{\mathbf{x}} \varepsilon_2(\mathbf{x}, a). \qquad (21)$$

Then the condition for total absolute error between $f(\mathbf{x})$ and $\phi(\mathbf{x})$ is derived from (18) and (21) by

$$|f(\mathbf{x}) - \phi(\mathbf{x})| \le \in(a) \qquad (22)$$

where $\in(a)$ is defined as $\in(a) = \in_1(a) + \in_2(a)$ with the property of $\lim_{a\to 0} \in(a) = 0$. Q.E.D.

One of examples using this type of function approximation is recovering functions from sampling. In the sampling theory, functions can be recovered by the following forms:

$$f(x) = \sum_i f(iX)Sinc[2B(x - iX)] \qquad (23)$$

$$f(x) = \sum_i f(iX)e^{-\pi[2B(x-iX)]} \qquad (24)$$

where X is a sampling distance defined by $X = \frac{1}{2B}$.

Note that (23) can perfectly recover the original function if the function is smooth (i.e., band-limited continuous function by the frequency domain of [0,B]) [14]. (24) implies the Gaussian low-pass filtering of the original function.

It is interesting to investigate the mapping capability of a network according to different types of kernel functions. The simulation results for the mapping capability indicate that the sinusoidal activation function provides the best mapping capability among the three, while the Gaussian activation function provides better mapping capability than the sigmoidal activation function [6]. Note, however, that in case the number of sample points is not large enough for the interpolation errors between samples to be ignored, the selection of an activation function should account for its capability of accurately interpolating the mapping between samples or its power of generalization. But the generalization power of an activation function may be highly dependent on the local characteristics of a particular mapping. This implies that the interpolation accuracy needs to be ensured adaptively through the self-recruitment of kernel functions based on training. In this case, an activation function which is not only powerful in generalizing a global mapping but also effective in refining local features without much altering the already learned mapping is desired. This makes a Gaussian activation function a good candidate for a network with self-recruitment. Here, a Gaussian activation function, an unnormalized form of Gaussian density function, is selected as an kernel function of the network, since the function is highly nonlinear, provides good locality for incremental learning, and has many well defined mathematical features. A Gaussian kernel function ψ_i is defined by

$$\psi_i = \psi(\mathbf{x}, \mathbf{p}_i) = e^{-d(\mathbf{x}, \mathbf{p}_i)/2} \qquad (25)$$

$$d(\mathbf{x}, \mathbf{p}_i) = d(\mathbf{x}, \mathbf{m}_i, \sigma_i) = \frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{\sigma_i^2} \qquad (26)$$

where $\mathbf{x}$ represents an input pattern, $\mathbf{m}_i$ and σ_i represent respectively the mean vector and the standard estimation of the ith Gaussian kernel function.

The network model proposed here is composed of three types of layers: the input layer, the hidden layer and the output layer. The input and output layers are composed of linear units, and the hidden layer is composed of Gaussian kernel functions. The weighted output values of the Gaussian kernel functions are summed by the connection between the hidden layer and the output layer in order to synthesize the desired function. Figure 1 illustrates the schematic diagram of the proposed network using Gaussian kernel function. Note that, for the network producing multiple outputs, we opt for each output being generated independently by its own set of kernel functions. This makes learning simpler.

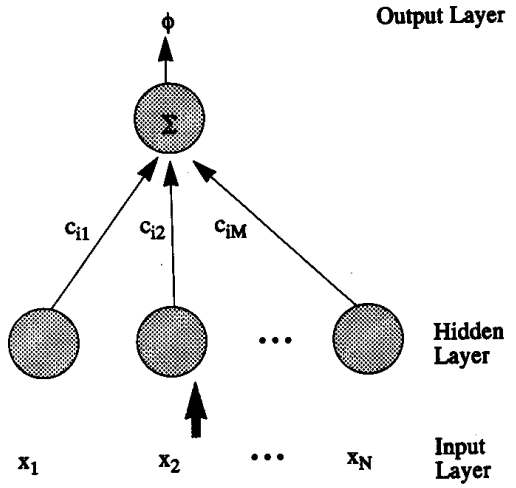


Fig. 1 The schematic diagram of a network using Gaussian kernel functions.

III. Parameter Estimation

For the proposed network, learning concerns mainly about the determination of minimally necessary number of kernel functions and the estimation of parameters of a network. The strategy to decide the minimally necessary number of kernel functions is to increase the number of kernel functions incrementally whenever a new kernel function needs to be defined for the further improvement of network performance, that is, reducing the network errors for the teaching patterns. The network performance can be related with the root mean square error defined by

$$E_{rms} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(y_k - \hat{y}_k(\mathbf{x}_k))^2} \qquad (27)$$

where N represents the number of training patterns, $(\mathbf{x}_k, y_k)$ represents (input, output) training patterns and $\hat{y}_k(\mathbf{x}_k)$represents an actual output of the network for the given input training pattern, $\mathbf{x}_k$.

For the proposed network, the following learning algorithm divided by three learning phases is suggested:

Phase 1 Learning: recruiting the necessary number of kernel functions

The goal of phase 1 Learning is to recruit the necessary number of kernel functions. In phase 1 learning, firstly, for the given input teaching pattern, the actual output of a network is generated and compared with the output teaching pattern. If the error between the actual output and the output teaching pattern is higher than specified error, a new kernel function is recruited at the position of the input teaching pattern and the shape of kernel function is decided according to the minimum distance between the position of a newly recruited kernel function and the positions of existing kernel functions. In this case, the output weights of kernel functions are adjusted in such a way that a network generates exact values with the output teaching patterns at the positions of kernel functions. For the hierarchical recruitment of kernel functions, the specified error is reduced as the number of iterations increases. The estimation of output weights for the exact mapping at the positions of kernel functions is determined as follows:

Let us first define a $k \times k$ matrix, $\mathbf{\Psi}_k$ as

$$\mathbf{\Psi}_k = \begin{bmatrix} \psi_{11} & \psi_{12} & \cdots & \psi_{1k} \\ \psi_{21} & \psi_{22} & \cdots & \psi_{2k} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \psi_{k1} & \psi_{k2} & \cdots & \psi_{kk} \end{bmatrix} \qquad (28)$$

where ψ_{ij} represents the output of the jth kernel function for the ith input teaching pattern.

Let us also define a k dimensional vector, $\mathbf{y}_k$ as $\mathbf{y}_k = [y_1, y_2, \cdots, y_k]^T$ where y_i represents the ith output teaching pattern. Then the output weight vector, $\boldsymbol{c}_k = [c_1, c_2, \cdots, c_k]^T$ where c_i represents the output weight of the ith kernel function, is given by

$$\mathbf{c}_k = \mathbf{\Psi}_k^{-1}\,\mathbf{y}_k \,. \qquad (29)$$

The k+1th matrix of (28) is given by

$$\mathbf{\Psi}_{k+1} = \begin{bmatrix} \mathbf{\Psi}_k & \vdots & \mathbf{u} \\ \cdots & \cdots & \cdots \\ \mathbf{v}^T & \vdots & \psi_{k+1\,k+1} \end{bmatrix} \qquad (30)$$

where $\mathbf{u}$ is a k dimensional vector defined by $\mathbf{u} = [\psi_{1,k+1}, \psi_{2,k+1}, \cdots, \psi_{k,k+1}]^T$ and $\mathbf{v}$ is a k dimen-

sional vector defined by $\mathbf{v} = [\psi_{1,k+1}, \psi_{2,k+1}, \cdots, \psi_{k,k+1}]^T$.

The inverse matrix of (30) can be represented by the following form:

$$\boldsymbol{\psi}_{k+1}^{-1} = \begin{bmatrix} \mathbf{A} & \vdots & \mathbf{b} \\ \cdots & \cdots & \cdots \\ \mathbf{d}^T & \vdots & c \end{bmatrix}. \quad (31)$$

Using the recursive formula of inverse matrix, **A**, **b**, c and **d** can be derived as follows:

$$\mathbf{A} = \boldsymbol{\psi}_k^{-1} + \frac{\boldsymbol{\psi}_k^{-1}\mathbf{u}\mathbf{v}^T\boldsymbol{\psi}_k^{-1}}{\psi_{k+1k+1} - \mathbf{v}^T\boldsymbol{\psi}_k^{-1}\mathbf{u}} \quad (32)$$

$$\mathbf{b} = -\frac{\boldsymbol{\psi}_k^{-1}\mathbf{u}}{\psi_{k+1k+1} - \mathbf{v}^T\boldsymbol{\psi}_k^{-1}\mathbf{u}} \quad (33)$$

$$c = +\frac{1}{\psi_{k+1k+1} - \mathbf{v}^T\boldsymbol{\psi}_k^{-1}\mathbf{u}} \quad (34)$$

$$\mathbf{d} = -\frac{\mathbf{v}^T\boldsymbol{\psi}_k^{-1}}{\psi_{k+1k+1} - \mathbf{v}^T\boldsymbol{\psi}_k^{-1}\mathbf{u}} \quad (35)$$

Since

$$\mathbf{c}_{k+1} = \boldsymbol{\psi}_{k+1}^{-1}\mathbf{y}_{k+1}, \quad (36)$$

$\mathbf{c}_{k+1} = [\mathbf{c}_k^{new}, c_{k+1}]^T$ can be evaluated as

$$\mathbf{c}_k^{new} = \mathbf{c}_k^{old} + \mathbf{b}e_{k+1} \quad (37)$$

$$\mathbf{c}_{k+1} = ce_{k+1} \quad (38)$$

where e_{k+1} represents the k+1th error defined by $e_{k+1} = y_{k+1} - \hat{y}_k + 1$ and $\hat{y}_k + 1$ represents actual output of the network for the k+1th teaching pattern.

Based on the derivation described above, the estimation procedure is summarized as follows:

Estimation Procedure (initial condition: $k = 0$, $n = 1$ and $\hat{y}_1 = 0$.)

Step 1 Present a new input pattern, $\mathbf{x}_n$ to the network.

Step 2 Get an actual output of the network, $\hat{y}_n = \phi(\mathbf{x}_n)$.

Step 3 Calculate an error of the network, $e_n = y_n - \hat{y}_n$

Step 4 Check the condition of recruitment:

If $e_n > e_c$ (error criteria),

- recruit a new kernel function such that

$$\mathbf{m}_{k+1} = \mathbf{x}_n \text{ and} \quad (39)$$

$$\sigma_{k+1} \propto \min_i ||\mathbf{x}_n - \mathbf{m}_i||, \quad (40)$$

- adjust $\mathbf{c}_{k+1}$ according to (37) and (38),
- and $k = k + 1$.

In the case of initial setting of σ, that is σ_1, any arbitrary value within the domain of input space can be selected.

Step 5 Estimate the rms error of the network:

- If one epoch of patterns are presented to the network, estimate the rms error of the network given by (27) where in this case, N represents the number of patterns in one epoch.
- Otherwise,
  - $n = n + 1$.
  - go to Step 1.

Step 6 Check the condition of termination:

- If E_{rms} > specified error,
  - $e_c = r_e e_c$ in which r_e represents the decrement rate of error criteria, e_c.
  - $n = 1$.

- go to Step 1.
- Otherwise, stop.

Phase 2 Learning: tuning of shape parameters of kernel functions

After phase 1 learning, the positions of kernel functions represent the reference points of the teaching patterns, i.e., the network generates the exact values at the positions of kernel functions. However, this is not so desirable from the view point of interpolation between the positions of kernel functions and of noisy teaching patterns. In this sense, the parameters (especially the shape parameters in this case) of kernel functions are adjusted in such a way to minimize the root mean square error between the desired and actual outputs so as to increase generalization capability. In phase 2 learning, the parameters of kernel functions are adjusted based on piecewise-linear approximation of a network in the space of the parameters of kernel functions. This type of parameter estimation is considered based on the assumption that the parameters of kernel functions are near the optimal values through phase 1 learning which gives rough approximation of the given teaching patterns. The estimation of parameter vector of kernel functions is determined as follows:

Firstly, let us assume a network with M kernel functions and define the kth teaching pattern, y_k as

$$y_k = \hat{y}_k(\boldsymbol{\sigma}) + w_k \tag{41}$$

where $\hat{y}_k(\boldsymbol{\sigma})$ represents the actual output of network in which $\boldsymbol{\sigma}$ represents the optimal parameter vector of kernel functions defined by $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, \cdots \sigma_M]^T$ and w_k represents white noise.

Applying the Taylor-series approximation to $\hat{y}_k(\boldsymbol{\sigma})$ around $\boldsymbol{\sigma}_0$,

$$\hat{y}_k(\boldsymbol{\sigma}) \simeq \hat{y}_k(\boldsymbol{\sigma}_0) + \left[\frac{\partial \hat{y}_k}{\partial \boldsymbol{\sigma}}(\boldsymbol{\sigma}_0)\right]^T (\boldsymbol{\sigma} - \boldsymbol{\sigma}_0). \tag{42}$$

Let us also define tilde $\tilde{y}_k$ as

$$\tilde{y}_k = y_k - \{\hat{y}_k(\boldsymbol{\sigma}) - \left[\frac{\partial \hat{y}_k}{\partial \boldsymbol{\sigma}}(\boldsymbol{\sigma}_0)\right]^T \boldsymbol{\sigma}_0\} \tag{43}$$

$$= \left[\frac{\partial \hat{y}_k}{\partial \boldsymbol{\sigma}}(\boldsymbol{\sigma}_0)\right]^T \boldsymbol{\sigma} + w_k \tag{44}$$

$$= \mathbf{h}_k^T \boldsymbol{\sigma} + w_k \tag{45}$$

where $\mathbf{h}_k$ is a vector defined by $\mathbf{h}_k = \frac{\partial \hat{y}_k}{\partial \boldsymbol{\sigma}}(\boldsymbol{\sigma}_0)$.

From (45), the optimal parameter vector, $\boldsymbol{\sigma}$ can be derived in the Linear Minimum Mean Squared Error (LMMSE) sense. Here, the resultant k+1th estimate of $\boldsymbol{\sigma}$, $\hat{\boldsymbol{\sigma}}_{k+1}$ is determined by

$$\hat{\boldsymbol{\sigma}}_{k+1} = \hat{\boldsymbol{\sigma}}_k + \mathbf{a}_k e_k \tag{46}$$

where e_k represents the network error for the kth teaching pattern defined by $e_k = y_k - \hat{y}_k(\hat{\boldsymbol{\sigma}}_k)$ and $\mathbf{a}_k$ is a vector defined by

$$\mathbf{a}_k = \mathbf{B}_k \mathbf{h}_k \text{ and} \tag{47}$$

$$\mathbf{B}_k = \mathbf{B}_{k-1} - \frac{\mathbf{B}_{k-1}\mathbf{h}_k\mathbf{h}_k^T\mathbf{B}_{k-1}}{1 + \mathbf{h}_k^T\mathbf{B}_{k-1}\mathbf{h}_k} \tag{48}$$

where $\mathbf{B}_0 = \frac{1}{\epsilon}\mathbf{I}$ and ϵ is a small constant. For large k, the choice of ϵ is unimportant.

Based on the derivation described above, the estimation procedure is summarized as follows:

Estimation Procedure (initial condition: k =1 and N_{epoch} = 0.)

Step 1 Present a new input pattern, $\mathbf{x}_k$ to the network.

Step 2 Get an actual output of the network, $\hat{y}_k = \phi(\mathbf{x}_k)$.

Step 3 Calculate an error of the network, $e_k = y_k - \hat{y}_k$.

Step 4 Update the parameter vector, σ_{k+1} of kernel functions according to (46).

Step 5 Check the condition of termination:

- If $k < N$,
  - $k = k+1$.
  - go to Step 1.
- Otherwise,
  - $N_{epoch} = N_{epoch} + 1$.
  - If N_{epoch} < specified number,
    - $k = 1$.
    - go to Step 1.

    Otherwise, stop.

Phase 3 Learning: tuning of output weights of a network

In phase 3 learning, the output weights of kernel functions are adjusted. This process is required since the basis vectors represented by the outputs of kernel functions are changed through phase 2 learning. The output weights of kernel functions are trained in the LMMSE sense. The estimation of output weights of kernel functions is determined as follows:

Let us define a vector $\mathbf{h}_k$ representing the output vector of M kernel functions as $\mathbf{h}_k = [\psi_{k1}, \psi_{k2}, \cdots, \psi_{kM}]^T$. Then similar to the phase 2 learning, the output weights of kernel functions can be derived in the LMMSE sense. Here, the resultant k+1th estimate of output vector, $\hat{c}_{k+1}$ is determined by

$$\hat{c}_{k+1} = \hat{c}_k + \mathbf{a}_k e_k \tag{49}$$

where e_k represents the network error for the kth teaching pattern defined by $e_k = y_k - \hat{y}_k(\hat{c}_k)$ and $\mathbf{a}_k$ is a vector defined by

$$\mathbf{a}_k = \mathbf{B}_k \mathbf{h}_k \text{ and} \tag{50}$$

$$\mathbf{B}_k = \mathbf{B}_{k-1} - \frac{\mathbf{B}_{k-1}\mathbf{h}_k\mathbf{h}_k^T\mathbf{B}_{k-1}}{1+\mathbf{h}_k^T\mathbf{B}_{k-1}\mathbf{h}_k} \tag{51}$$

where $\mathbf{B}_0 = \frac{1}{\epsilon}\mathbf{I}$ and ϵ is a small constant. For large k, the choice of ϵ is unimportant.

Based on the derivation described above, the estimation procedure is summarized as follows:

Estimation Procedure (initial condition: $k = 1$ and $N_{epoch} = 0$.)

Step 1 Present a new input pattern, $\mathbf{x}_k$ to the network.

Step 2 Get an actual output of the network, $\hat{y}_k = \phi(\mathbf{x}_k)$.

Step 3 Calculate an error of the network, $e_k = y_k - \hat{y}_k$.

Step 4 Update the output weights, $\mathbf{c}_{k+1}$ of kernel functions according to (49).

Step 5 Check the condition of termination:

- If $k < N$,
  - $k = k+1$.
  - go to Step 1.
- Otherwise,
  - $N_{epoch} = N_{epoch} + 1$.
  - If N_{epoch} < specified number,
    - $k = 1$.
    - go to Step 1.

    Otherwise, stop.

The whole learning sequence is going through phase 1, 2 and 3 learning processes. Here, phase 1 learning process is related to the recruitment of Gaussian kernel functions while phase 2 and 3

learning processes are related to the parameter estimation. The number of iterations of phase 1 learning can be decided by user's requirement on the desired level of accuracy of the network. For the phase 2 and 3 learning processes, it usually takes 2 or 3 epochs to achieve the near local optimal parameters. The phase 2 and 3 learning processes can be continuously performed in an alternate way for the further minimization of network error. This helps to find the near global optimal parameters. However, in most cases, 1 pass of phase 1, 2 and 3 learning processes is sufficient in the sense of minimizing both computation time and rms error of the network. The convergence of phase 2 and 3 learnings are guaranteed assuming that the nonlinear parameters of Gaussian kernel functions are well approximated by the first order Taylor expansion. This assumption is quite valid since the network approximates the given teaching patterns in such a way to have exact values at the positions of Gaussian kernel functions during phase 1 learning process and this implies that only minor adjustment (fine tuning) of network parameters is required for the interpolation between the positions of Gaussian kernel functions.

IV. Simulation

The suggested learning algorithm is applied to the various cases of function approximation. As the examples of function approximation, four sets of data are selected: 2 sets of data for one dimensional continuous functions, 1 set of data for two dimensional continuous function and 1 set of data for Mackey-Glass chaotic time-series.

For the simulation of one dimensional continuous functions, linear and sinusoidal functions as illustrated in Figures 2-(a) and (b) respectively, are considered. The given equations are

$$f_1(x) = 2|x| \text{ and} \tag{52}$$

$$f_2(x) = sin(\pi x) \text{ for } |x| < 1. \tag{53}$$

For each function, 500 teaching patterns are randomly selected, another randomly selected 500 patterns are used for the test patterns to evaluate the performance of training. With 20 epochs of phase 1 learning and 2 epochs of phase 2 and 3 learning, 20 and 13 kernel functions are recruited for linear and sinusoidal functions respectively. The rms errors for linear and sinusoidal functions are evaluated as 0.0030 and 0.0026 respectively. The error curves for linear and sinusoidal functions are illustrated in Figures 2-(c) and (d) respectively.

For the simulation of two dimensional continuous function, the following sinusoidal function is considered:

$$f_3(x, y) = 0.4\sin(\pi x) + 0.6\cos(\pi y) \tag{54}$$

where the domains of x and y are restricted by the values between -1 and 1.

The 3-D mesh graph of $f(x, y)$ is shown in Figure 3-(a). For the teaching patterns, 1000 patterns are randomly selected from the given function, and another 1000 randomly selected patterns are used for the test patterns to evaluate the performance of training. The actual output of a network after 15 epochs of phase 1 learning and 2 epochs of phase 2 and 3 learning, is illustrated in Figure 3-(b). The the error surface after learning is illustrated in Figure 3-(c). In this error curve, the error values near the boundary of input space domain appear high value since relatively smaller number of teaching patterns are trained in this region. The rms error of this training is evaluated as 0.0155. For this level of performance, 34 kernel functions are recruited. In [15], 18 Gaussian units are

recruited to achieve the similar level of performance. However, in this approach, it takes more than 100 times of processing time.

The discrete version of the *Mackey-Glass (M-G)* chaotic time-series [16] is considered as an another example of function approximation. The discrete version of the M-G time-series is described by

$$x(t+1) = (1-a)x(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)}. \qquad (55)$$

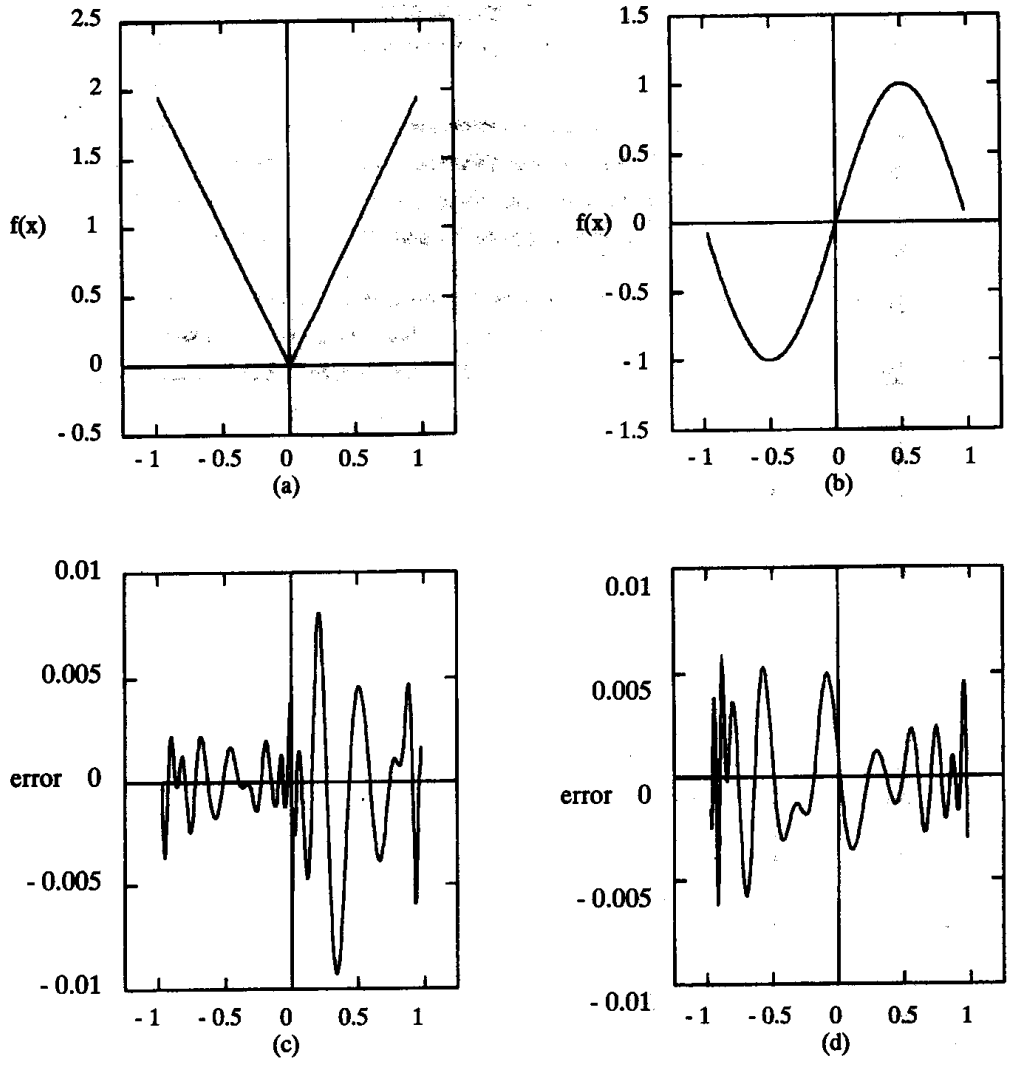


Fig. 2. Simulation Results for 1-D data : (a) and (c) represent the linear type teaching function and error between the teaching function and the actual output of network respectively. (b) and (d) represent the sinusoidal teaching function and error between the teaching function and the actual output of network respectively.

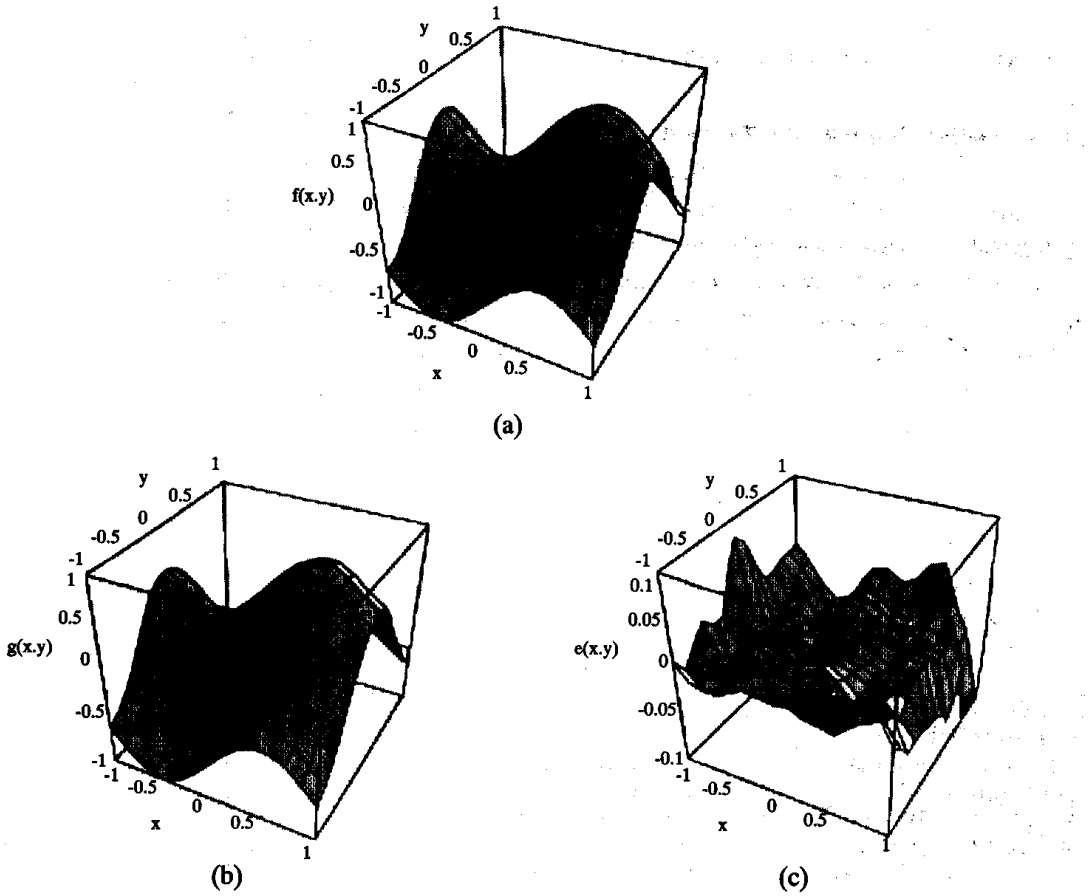


Fig. 3. Simulation Results for 2-D data : (a), (b) and (c) represent the 3-D mesh graphs for the teaching function, the output of networkafter learning and the error between the teaching function and the output of network respectively.

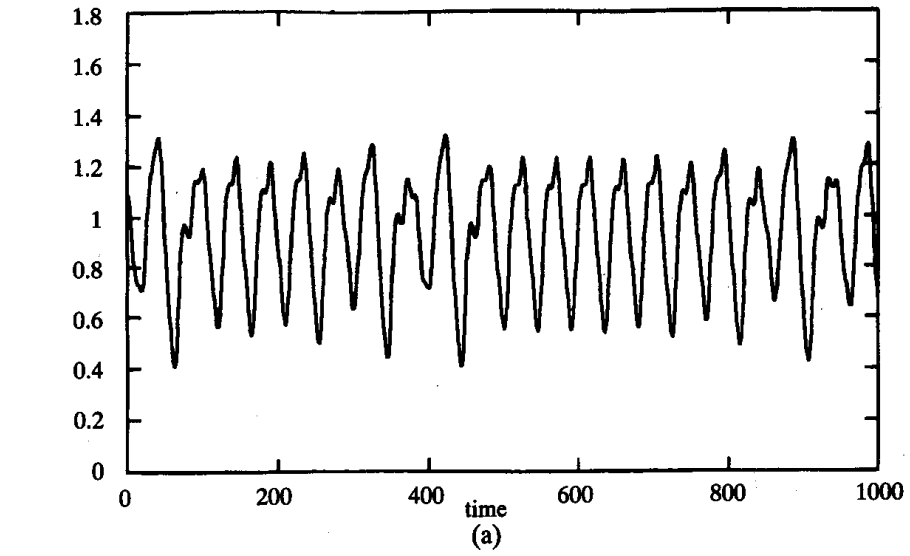
By setting a = 0.1, b = 0.2, and τ =17, a chaotic time series with a strange attractor is produced [17]. The following form of time-series,

$$x(t+85)= f\big(x(t), x(t-6), x(t-12), x(t-18)\big) \quad (56)$$

is used for the estimation of M-G chaotic time-series. Similar to the previous works [16, 17], the suggested network were trained with the 500 training data and were tested with the succeeding 500 data. The generated M-G chaotic time-series is shown in Figure 4-(a). To define the prediction accuracy, the normalized root mean squared error[1] is considered to remove the dependency on the dynamic range of data.

The results of simulation are shown Figure 4-(b): the curves shown in Figure 4-(b) illustrate the

[1] The normalized root mean squared error defined by the root mean squared error divided by the standard deviation of the given time-series, x (·).



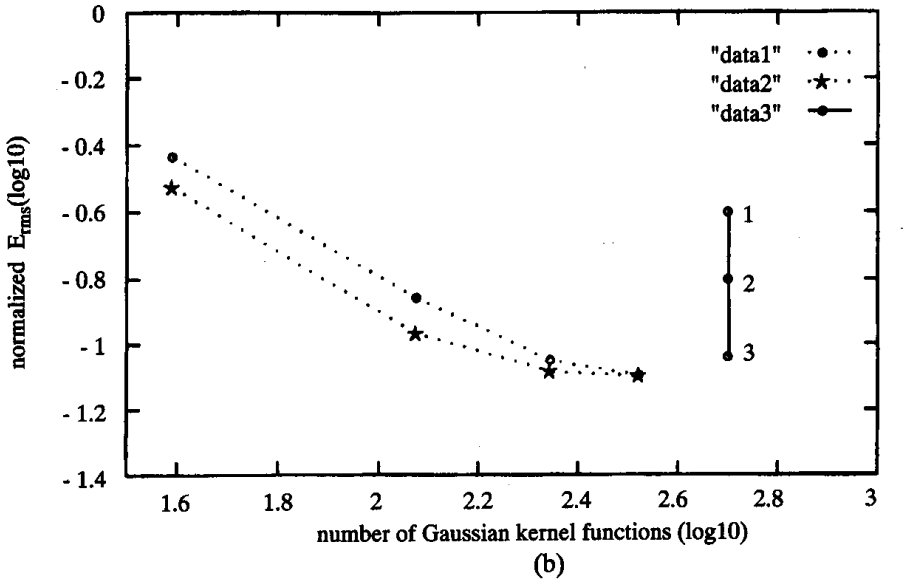


Fig. 4. Simulation Results for Mackey-Glass chaotic time-series : (a) represents the tested Mackey-Glass chaotic time-series and (b) represents the plotting of normalized rms error versus the number of kernel functions. In (b), data1 and data2 represent the prediction accuracy with phase 1 learning process, and phase 1, 2 and 3 learning processes respectively while the number in data3 represents the prediction accuracy for three different learning methods suggested by Moody and Darken.

prediction accuracy versus the number of kernel functions for the testing data. Here, data1 and data2 represent the prediction accuracy with phase 1 learning process, and phase 1, 2 and 3 learning processes respectively while the number in data3 represents the prediction accuracy for three different learning methods suggested by Moody and Darken [8]. In data3, 500 teaching patterns are used for 1 and 2 while 5000 teaching patterns are used for 3.

The simulation results represented by data1 and data2 show us that 1) the normalized errors for the training data can be reduced to smaller values by increasing the number of kernel functions, but 2) those for the testing data begin to level off around 300 kernel functions. This indicates that 500 training data are not sufficient for the network to generalize the testing data. Note also that the parameter adaptation processes (phase 2 and 3 learning processes) are not very effective on reducing the prediction error when the number of Gaussian kernel functions becomes larger. This is due to the fact that the degree of freedom at which the network parameters can be varied becomes smaller when the number of Gaussian kernel functions becomes larger. In the case that our approach is compared data3 made by [8], our approach requires smaller number of kernel functions and teaching patterns to achieve similar level of performance: for instance, when compared best data of [8] (refer 3 on data3), around 2 times more processing units and 10 times more training data are required for [8] to achieve the similar level of prediction accuracy.

V. Conclusion

A new way of nonparametric estimation for the function approximation is presented in this paper. The proposed network of estimation model is composed of input, hidden and output layers in which the input and output layers have linear activation units while the hidden layer has nonlinear activation units or *kernel functions* which have the characteristics of bounds and locality.

For the proposed network, learning concerns mainly about the determination of minimally necessary number of kernel functions and the estimation of parameters of a network. The strategy to decide the minimally necessary number of kernel functions is to increase the number of kernel functions incrementally whenever a new kernel function needs to be defined for the further improvement of network performance. For the parameter estimation of a network, linear learning rule is applied between hidden and output layers while nonlinear (piecewise-linear) learning rule is applied between input and hidden layers. The linear learning rule updates the output weights between hidden and output layers based on the Linear Minimization of Mean Square Error (LMMSE) sense in the space of kernel functions while the nonlinear learning rule updates the parameters of kernel functions based on the gradient of mean square error with respect to the parameters (especially, the shape) of kernel functions. This approach of parameter adaptation provides near optimal values of the parameters associated with kernel functions in the sense of minimizing mean square error. The simulation results for the function approximation show that the suggested nonparametric estimation is efficient from the view point of the number of kernel functions as well as learning speed.

The suggested network can contribute to the advancement of a new methodology for designing a more general form of mapping networks. The obvious advantage of the suggested estimation technique is providing a solution to the problems

encountered in conventional learning techniques due to the existence of local minima, flat error surface curvature, as well as structural inflexibility.

Acknowledgements

This research was partly funded by Korea Telecom. The author would like to appreciate Professor Sukhan Lee at USC for his helpful inspiration. The author also would like to thank Dr. El Hang Lee for his cooperation on this project.

References

[1] G. A. Carpenter and S. Grossberg, "Art2: Stable self-organization of pattern recognition codes for analog input patterns," *Applied Optics*, 26:4919~4930, 1987.

[2] D. L. Reilly, L. N. Cooper, and C. Elbaum, "A neural model for category learning," *Biological Cybernetics*, 45:35~41, 1982.

[3] R. Hecht-Nielsen, "Kolmogorov mapping neural network existence theorem," *IEEE International Conference on Neural Networks*, 3:11~13, 1987.

[4] M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning," *Avtomatika i Telemekhanika*, 25:917~936, 1964.

[5] S. Lee and R. M. Kil. "Multilayer feedforward potential function network," *IEEE International Conference on Neural Networks*, 1:161~171, 1988.

[6] S. Lee and R. M. Kil. "A gaussian potential function network with hierarchically self-organizing learning," *Neural Networks*, 4(2):207~224, 1991.

[7] M. Niranjan and F. Fallside, Neural networks and radial basis functions in classifying static speech patterns. Technical Report CUED/F-INFENG/TR22, Cambridge University, 1988.

[8] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, 1:281~294, 1989.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Parallel Distributed Processing*, volume 1, pages 318~362. MIT Press/Bradford Books, 1986.

[10] M. Minsky and S. Papert, *Perceptrons*. The MIT Press, 1969.

[11] E. Parzen, "On the estimation of a probability density function and mode," *Annals of Mathematical Statistics*, 33:1065~1076, 1962.

[12] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, 2(3):183~192, 1989.

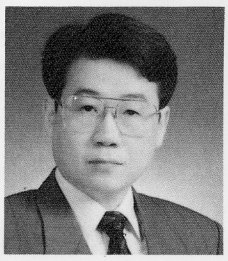
[13] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, 2:359~366, 1989.

[14] C. E. Shannon, "Communication in the presence of noise," *Proceedings of IRE*, 37:10, 1945.

[15] S. Lee and R. M. Kil, "Nonlinear system control based on gaussian potential function network," *IEEE International Symposium on Intelligent Control*, pages 423~429, 1991.

[16] A. S. Lapedes and R. Farber, Nonlinear signal processing using neural networks: Prediction and system modeling, Technical Report LA-UR-87-2662, Los Alamos National Laboratory, 1987.

[17] A. S. Lapedes and R. Farber, "How neural nets work," *Neural Information Processing Systems*, pages 442~456, 1988.



Rhee M. Kil received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea in 1979 and the M.S. and Ph.D. degrees in computer engineering from the University of Southern California, Los Angeles, U.S.A. in 1985 and 1991, respectively.

From 1979 to 1983 he was with Agency for Defense Development, Daejeon, Korea where he was involved in the development of Infrared Imaging System. From 1987 to 1991 his research topic was concentrated on the theories and applications of connectionist models. His dissertation was on the learning algorithms of connectionist models and their applications to the nonlinear system control. Since 1991, he has been with Research Department of ETRI, Daejeon, Korea pursuing interests in speech recognition, time series prediction, data coding and nonlinear system control.