Neural Networks, Vol. 4, pp. 207–224, 1991
Printed in the USA. All rights reserved.

0893-6080/91 $3.00 + .00

ORIGINAL CONTRIBUTION

A Gaussian Potential Function Network With Hierarchically Self-Organizing Learning

SUKHAN LEE AND RHEE M. KIL

University of Southern California, Los Angeles

(*Received* 29 *June* 1988; *revised and accepted* 25 *October* 1990)

Abstract—*This article presents a design principle of a neural network using Gaussian activation functions, referred to as a Gaussian Potential Function Network (GPFN), and explores the capability of a GPFN in learning a continuous input–output mapping from a given set of teaching patterns. The design principle is highlighted by a Hierarchically Self-Organizing Learning (HSOL) algorithm featuring the automatic recruitment of hidden units under the paradigm of hierarchical learning.*

A GPFN generates an arbitrary shape of a potential field over the domain of the input space, as an input–output mapping, by synthesizing a number of Gaussian potential functions provided by individual hidden units referred to as Gaussian Potential Function Units (GPFUs). The construction of a GPFN is carried out by the HSOL algorithm which incrementally recruits the minimum necessary number of GPFUs based on the control of the effective radii of individual GPFUs, and trains the locations (mean vectors) and shapes (variances) of individual Gaussian potential functions, as well as their summation weights, based on the Backpropagation algorithm.

Simulations were conducted for the demonstration and evaluation of the GPFNs constructed based on the HSOL algorithm for several sets of teaching patterns.

Keywords—Mapping neural network, Multilayer feedforward network, Gaussian potential function, Nonparametric estimation, Self-organizing learning, Accommodation boundary, Effective radius.

1. 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 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 (Carpenter & Grossberg, 1987), and to the identification of class boundaries (Reilly, Cooper, & Elbaum, 1982), a major effort needs to be expended to establish a learning algorithm capable of automatically configuring a network based on the self-determination of network architecture and the self-recruitment of neurons with a proper type of activation functions. This article initiates such an effort by considering a neural network with nonsigmoidal activation functions and the capability of self-recruiting neurons under the paradigm of hierarchical learning.

Let us first define a Mapping Neural Network (MNN) (Hecht-Nielsen, 1987b) as a network performing a mapping, ϕ, from a compact set, I^n (for example, an n dimensional Euclidean unit cube) to an m dimensional Euclidean space, R^m, $\phi: I^n \rightarrow R^m$, based on the interconnection of neurons as basic non-

This work is supported by the Faculty Research and Innovation Fund of University of Southern California. Stimulation and encouragement by Dr. Kosko of University of Southern California is appreciated.

Requests for reprints should be sent to S. Lee, Department of Electrical Engineering—Systems, University of Southern California, Los Angeles, CA 90089.

linear computational units in a parallel and distributed manner. Then, most of the currently existing artificial neural networks can be interpreted within the framework of a MNN:

1. An *associative memory* such as the Autoassociative Memory (AM) (Hopfield, 1982, 1984), the Bidirectional Associative Memory (BAM) (Kosko, 1987), or the Boltzmann Machine (Ackley, Hinton, & Sejnowski, 1985), stores an input–output mapping in a content addressable memory where the network converges to a stored memory, based on the dynamics imbedded in the network structure.
2. A neural network based on the *competitive network* such as the Hamming Network (Lippmann, 1987), the Self-Organizing Feature Map (Kohonen, 1984), the Art2 (Carpenter & Grossberg, 1987), the Counterpropagation Network (Hecht-Nielsen, 1987a) or the Neural Model for Category Learning (Reilly et al., 1982), selects a neuron through the winner-takes-all competition based on the closeness between the reference patterns that individual neurons represent and the given input pattern, and generates an output as the reference pattern of the selected neuron.
3. The *multilayer feedforward network* such as the Backpropagation Network (Rummelhart, Hinton, & Williams, 1986), the Neocognitron (Fukushima, 1980, 1988) or the Athena (Koustogeras & Papachritou, 1988), (approximately) realizes an arbitrary input–output mapping or a decision boundary by transforming the input domain into increasingly complex nonlinear manifolds in the spaces of upper layers through a series of intermediate mappings.

We are interested in analyzing the capability of an MNN in realizing an arbitrary function. The Kolmogorov/Sprecher theorem (see Appendix A; Kolmogorov, 1957; Sprecher, 1965) indicates that any continuous real function can be exactly realized by a four-layer neural network composed of an input, an output, and two hidden layers with a finite number of neurons (Hecht-Nielsen, 1987b). However, since no constructive method for determining the activation function of hidden layers is currently available, a direct application of the Kolmogorov/Sprecher theorem to the realization of an MNN is not feasible. Irie and Miyake (1988) proved that a three-layer network with an infinite number of hidden units can represent an arbitrary function, provided that the activation functions of hidden units as well as the mapping functions are bounded and absolutely integrable. Funahashi (1989) extended the Irie–Miyake theorem to include sigmoidal activation functions, such that any continuous function is approximately realizable by a three-layer network with hidden units having, monotonically increasing but bounded, continuous activation functions. A similar result was obtained by Hecht-Nielsen (1989), where it was shown that a subset of a backpropagation network can implement a sinusoidal function, enabling the backpropagation network to perform the Fourier series approximation of an arbitrary function. Hornik, Stinchcombe, and White (1989) also proved that a three-layer network with a sufficiently large number of hidden units having an arbitrary squashing activation functions can be used as a universal function approximator.

The aforementioned theorems concern the capability of an MNN as a universal function approximator. However, the following research issues should be raised and investigated:

1. What are the effects of the number of layers on the realization and training of an MNN? Does an increase in the number of hidden layers render an MNN more efficient in terms of the number of neurons or in terms of the learning speed in training?
2. What are the effects of a nonsigmoidal activation function on the realization and training of an MNN? Note that any absolutely integrable activation function, such as a Gaussian function, is capable of approximating an arbitrary function, and that a sinusoidal activation function can be used for approximating an arbitrary function based on the Fourier series expansion. Can nonsigmoidal activation functions such as Gaussian and sinusoidal functions provide a better alternative to the sigmoidal activation function in terms of realizing and training an MNN due to their nonlinear mapping capabilities?

In what follows, we will show that nonsigmoidal activation functions provide the network with the capability of forming more complex nonlinear manifolds of the input domain, which is equivalent to what multiple hidden layers can provide through intermediate mappings.

This article presents a nonsigmoidal MNN, called the Gaussian Potential Function Network (GPFN) (Lee & Kil, 1988) and investigates the capability of the GPFN in realizing an arbitary mapping. The GPFN 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 Gaussian computational units called Gaussian Potential Function Units (GPFUs). 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). The distinctive feature of HSOL is its capability of automatically re-

cruiting necessary GPFUs under the paradigm of hierarchical learning, implemented through the successive adjustment of the accommodation boundaries or the effective radii of individual GPFUs in the input domain.

2. POTENTIAL FUNCTION NETWORK

2.1. Design Concept

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* (Aizerman, Braverman, & Rozonoer, 1964) as follows:

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

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 $\|\mathbf{x} - \mathbf{x}_i\|$, that is, $K(\mathbf{x}, \mathbf{x}_i)$ has the maximum value at $\mathbf{x} = \mathbf{x}_i$ and decreases monotonically to zero as $\|\mathbf{x} - \mathbf{x}_i\|$ approaches infinity.

A learning algorithm similar to that of the Perceptron (Minsky & Papert, 1969) has been proposed for applying eqn (1) to binary classification:

$$\phi^{\text{new}}(\mathbf{x}) = \begin{cases} \phi^{\text{old}}(\mathbf{x}) + K(\mathbf{x}, \mathbf{x}_k) & \text{if the sample, } \mathbf{x}_k \text{ is labelled} \\ & +1 \text{ and } \phi^{\text{old}}(\mathbf{x}_k) \le 0 \\ \phi^{\text{old}}(\mathbf{x}) - K(\mathbf{x}, \mathbf{x}_k) & \text{if the sample, } \mathbf{x}_k \text{ is labelled} \\ & -1 \text{ and } \phi^{\text{old}}(\mathbf{x}_k) \ge 0 \\ \phi^{\text{old}}(\mathbf{x}) & \text{otherwise} \end{cases} \tag{2}$$

It has been shown that eqn (2) converges within finite steps.

The potential function approach to binary classification described by eqns (1) and (2) has a similar flavor to the nonparametric estimation of a probability density function based on the *Parzen window* (Parzen, 1962). 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

$$p_n(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h_n^d} \psi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_n}\right) \tag{3}$$

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 $\lim_{n\to\infty} nh_n^d = \infty$. It can be shown from eqn (3) that $p_n(\mathbf{x})$ converges to $p(\mathbf{x})$ as n approaches ∞.

The problem associated with eqn (1) or (3) 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 eqn (1) or (3) are 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 eqn (1) or (3): the position shift of a potential function should correspond to the coordinate of input samples and the shape of a potential function should be fixed, and by introducing a methodology of self-recruiting a minimum necessary number of potential functions with the capability of adjusting both the position shift and the shape parameters of individual potential functions.

A generalized form of eqn (1) or (3), incorporating the adjustment of shape parameters and the self-recruitment of potential functions, can be expressed as

$$\phi(\mathbf{x}) \equiv \sum_{i=1}^{M} c_i \psi(\mathbf{x}, \mathbf{p}_i) \tag{4}$$

where M represents the number of potential 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 potential function. In eqn (4), M, c_i, and $\mathbf{p}_i$, $i = 1, \cdots, M$, are subject to the adjustment through learning. Eqn (4) may be able to achieve a desirable error level in function approximation with a smaller number of potential functions, but may require a more complicated learning algorithm. Thus, this article focuses on developing a learning algorithm for self-recruiting a minimum necessary number of potential functions, M, and for training the shape parameters as well as summation weights, $\mathbf{p}_i$ and c_i, $i = 1, \cdots, M$.

Prior to the exploitation of such a learning algorithm, let us first investigate the effect of selecting a different type of activation functions (among sigmoidal, Gaussian, and sinusoidal) on the power of function approximation based on eqn (4). Note that, in eqn (4), an output vector of the input layer, $\mathbf{x}$, is directly used as an input vector to a hidden unit. This can be compared with conventional three-layer feedforward network in which the weighted summation of individual input elements, $\sum_{j=1}^{I} \lambda_{ij} x_j$, is fed into a hidden unit:

$$\phi(\mathbf{x}) = \sum_{i=1}^{M} c_i \psi\left(\sum_{j=1}^{I} \lambda_{ij} x_j, \mathbf{p}_i'\right). \tag{5}$$

Equation (4) represents a more general form of a multilayer feedforward network, where a multidimensional output vector from the layer $i - 1$ is directly used as an input vector to the layer i, assuming that a neuron is capable of processing a vector. This

implies that the network has additional freedom of selecting how to process a multidimensional input vector inside a neuron. In this sense, eqn (5) is considered as a special case of eqn (4).

According to Funahashi (1989) and Hornik et al. (1989), eqn (4) can approximate an arbitrary 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. To further investigate the capability of eqn (4) in realizing an arbitrary function in relation to the number as well as the shape of PFUs, let us represent $\psi(\mathbf{x})$ based on N discrete sample points: $\phi(\mathbf{x}_1)$, $\phi(\mathbf{x}_2)$, $\cdots$, $\phi(\mathbf{x}_N)$. Then, with the following definitions,

$$\mathbf{z} \equiv [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \cdots, \phi(\mathbf{x}_N)]^t \tag{6}$$

$$\mathbf{c} \equiv [c_1, c_2, \cdots, c_M]^t \text{ and} \tag{7}$$

$$\mathbf{y}_i \equiv [\psi(\mathbf{x}_1, \mathbf{p}_i), \psi(\mathbf{x}_2, \mathbf{p}_i), \cdots, \psi(\mathbf{x}_N, \mathbf{p}_i)]^t \text{ for } i = 1, \cdots, M \tag{8}$$

we can obtain the discrete form of eqn (4), as follows:

$$\mathbf{z} = \mathbf{Y}\mathbf{c} \tag{9}$$

where $\mathbf{Y} \equiv [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M]: N \times M$ matrix.

The question is whether and how, for a given $\mathbf{z}$, we can adjust $\mathbf{Y}$ to find an exact solution for $\mathbf{c}$ satisfying eqn (9) or an optimal solution minimizing the error, E,

$$E \equiv \|\mathbf{z} - \mathbf{Y}\mathbf{c}\|^2 \tag{10}$$

In case $M \geq N$, there always exists one or more exact solutions for $\mathbf{c}$ that satisfy eqn (9). However, the condition $M \geq N$ is unrealistic since N should be selected very large for the minimization of interpolation errors. In case $M < N$, eqn (4) represents an overdetermined set of equations, and thus, the existence of the exact solution for $\mathbf{c}$ depends on the special condition imposed on $\mathbf{Y}$, that is, rank $[\mathbf{Y}:\mathbf{z}] = M$. Whether $\mathbf{Y}$ can be set to satisfy the condition, rank$[\mathbf{Y}:\mathbf{z}] = M$, by adjusting the parameters of individual PFUs, is a problem which needs to be explored.

The condition, rank$[\mathbf{Y}:\mathbf{z}] = M$, is equivalent to the condition that $\mathbf{z}$ is embedded in the subspace, S_Y, spanned by $\mathbf{y}_1, \cdots, \mathbf{y}_M$, where $\mathbf{z}, \mathbf{y}_i, i = 1, \cdots, M$ are defined in the N dimensional sample space, S_N. Note that with fixed $\mathbf{y}_i, i = 1, \cdots, M$, $\mathbf{z} \in S_Y$ implies that no exact solution for $\mathbf{c}$ exists; instead, the optimal solution, $\mathbf{c}^*$, which minimizes the error, eqn (10), can be obtained by projecting $\mathbf{z}$ onto S_Y such that

$$\mathbf{c}^* = \mathbf{Y}^+\mathbf{z} \tag{11}$$

$$E_{\min} = \|(\mathbf{I} - \mathbf{P})\mathbf{z}\|^2 \tag{12}$$

where $\mathbf{Y}^+$ represents the generalized inverse of $\mathbf{Y}$, $\mathbf{Y}^+ \equiv (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t$, and $\mathbf{P}$ presents the projection matrix, $\mathbf{P} \equiv \mathbf{Y}\mathbf{Y}^+$.

The adjustment of $\mathbf{p}_i$ may provide the setting of $\mathbf{y}_i, i = 1, \cdots, M$ that makes $\mathbf{z}$ embed in S_Y. The adjustment of $\mathbf{p}_i$ of $\psi(\mathbf{x}, \mathbf{p}_i)$ generates the trajectory or the range of $\mathbf{y}_i$, $\mathcal{R}(\mathbf{y}_i)$ in S_N according to $\mathbf{y}_i = [\psi(\mathbf{x}_1, \mathbf{p}_i), \psi(\mathbf{x}_2, \mathbf{p}_i), \cdots, \psi(\mathbf{x}_N, \mathbf{p}_i)]^t$ with fixed sample points $\mathbf{x}_k, k = 1, \cdots, N$. Note that the trajectory of $\mathbf{y}_i$, $\mathcal{R}(\mathbf{y}_i)$, is the same for all $i, i = 1, \cdots, M$, assuming that the activation functions, $\psi(\mathbf{x}, \mathbf{p}_i), i = 1, \cdots, M$, are of the same mathematical form. With M PFUs, we can arbitrarily select M points from $\mathcal{R}(\mathbf{y}_i)$ to form M vectors, $\mathbf{y}_i^k, i = 1, \cdots, M$ of $\mathbf{Y}^k$. The linear combination of the selected M vectors defines a linear manifold, $\mathcal{L}(\mathbf{Y}^k)$, in S_N. Then the following theorem holds:

Theorem 1: $\mathbf{z} \equiv [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \cdots, \phi(\mathbf{x}_N)]^t$ *is exactly realizable iff* $\mathbf{z} \in \cup_{k \in K} \mathcal{L}(\mathbf{Y}^k)$, *where* K *is a finite or infinite index set representing all the possible selections of* M *points from* $\mathcal{R}(\mathbf{y}_i)$.

Proof: $\mathbf{z} \in \cup_{k \in K} \mathcal{L}(\mathbf{Y}^k)$ implies that there exists $l \in K$ such that $\mathbf{z} = \mathbf{Y}^l\mathbf{c}$. Q.E.D.

The implication of the above theorem is as follows:

1. $\cup_{k \in K} \mathcal{L}(\mathbf{Y}^k)$ provides a measure indicating the mapping capability of a neural network in realizing an arbitrary function with a finite number of hidden units, since it specifies the range of an exactly realizable $\mathbf{z}$, or the collection of functions corresponding to such an exactly realizable $\mathbf{z}$. $\cup_{k \in K} \mathcal{L}(\mathbf{Y}^k) = S_N$ implies that any arbitrary $\mathbf{z}$ can be realizable.
2. Given $\mathbf{z}$, or the range of $\mathbf{z}$, we may select the type of an activation function (or possibly, a combination of different types of activation functions), for example, among sigmoid, Gaussian and sinusoidal functions, that is most suitable for the exact realization of $\mathbf{z}: \mathbf{z} \in \cup_{k \in K} \mathcal{L}(\mathbf{Y}^k)$.

Figure 1 illustrates the above concepts based on a simple network with two PFUs, $\psi(\mathbf{x}, \mathbf{p}_1)$ and $\psi(\mathbf{x}, \mathbf{p}_2)$, and three training samples $\phi(\mathbf{x}_1)$, $\phi(\mathbf{x}_2)$, $\phi(\mathbf{x}_3)$, investigating the realization of $\mathbf{z}$, $\mathbf{z} = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \phi(\mathbf{x}_3)]^t$, with $\mathbf{y}_1$, $\mathbf{y}_1 = [\psi(\mathbf{x}_1, \mathbf{p}_1), \psi(\mathbf{x}_2, \mathbf{p}_1), \psi(\mathbf{x}_3, \mathbf{p}_1)]^t$, and $\mathbf{y}_2$, $\mathbf{y}_2 = [\psi(\mathbf{x}_1, \mathbf{p}_2), \psi(\mathbf{x}_2, \mathbf{p}_2), \psi(\mathbf{x}_3, \mathbf{p}_2)]^t$, based on $\mathbf{z} = [\mathbf{y}_1, \mathbf{y}_2]\mathbf{c}$. The measure of mapping capability, $\cup_{k \in K} \mathcal{L}(\mathbf{Y}^k)$, is obtained by simulation for the case of sigmoidal, Gaussian and sinusoidal PFUs (with single shape parameter), and illustrated respectively in Figures 1(b), (c), and (d). In each figure, the dotted area represents $\cup_{k \in K} \mathcal{L}(\mathbf{Y}^k)$. The simulation results indicate that the sinusoidal activation function provides the best mapping capability among the three, while the Gaussian activation function pro-

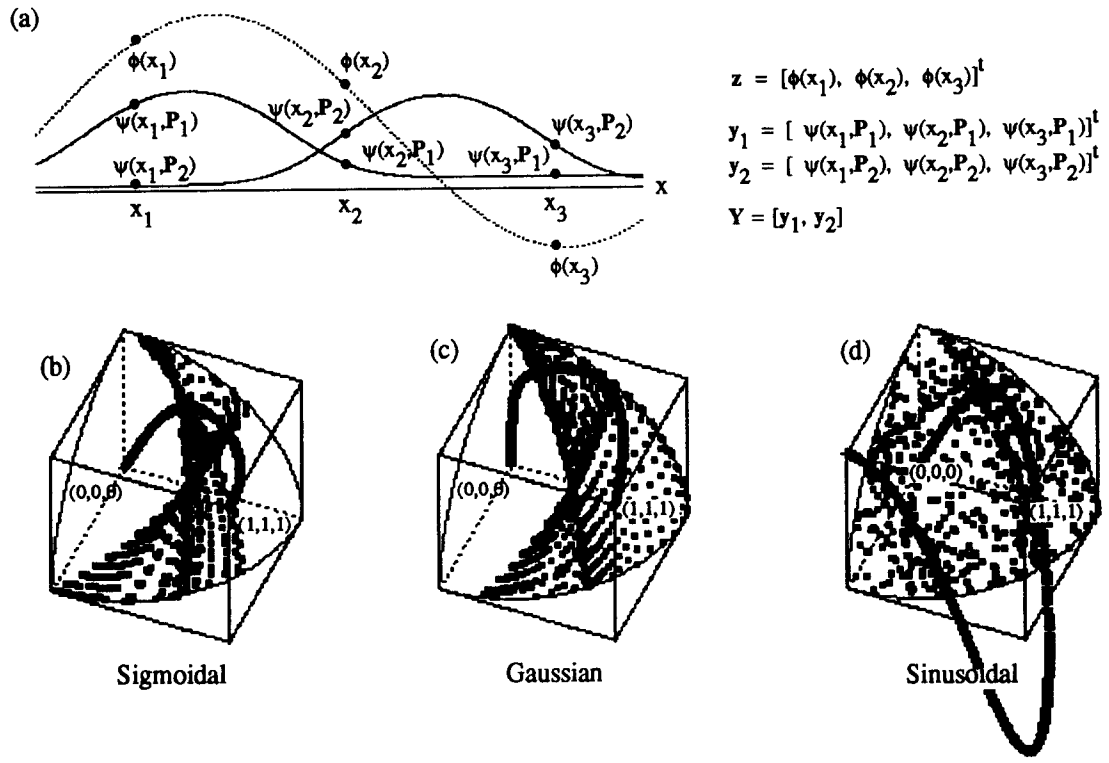


FIGURE 1. An example to illustrate $\mathcal{R}(y_i)$ and $\cup_{k\in K}\mathcal{L}(Y^k)$ for sigmoidal $((1 - e^{-x/a})/(1 + e^{-x/a}))$, Gaussian $(e^{-x^2/2a^2})$, and sinusoidal $(sin(ax))$ PFUs. (a) Represents an example with 2 PFUs, $\psi(x, p_1)$ and $\psi(x, p_2)$, and 3 samples, $\phi(x_1)$, $\phi(x_2)$ and $\phi(x_3)$. In (b), (c) and (d), $\mathcal{R}(y_i)$ is represented by a thick line contour on the cube, $[0, 1]^3$ of sample space, while $\cup_{k\in K}\mathcal{L}(Y^k)$ is represented by the dotted area on the surface of the sphere fitted inside the cube. The volume defined by the dotted area and the origin through a solid angle represents $\cup_{k\in K}\mathcal{L}(Y^k)$ inside the $\frac{1}{8}$ sphere. The dotted areas were obtained by the trajectories of great circles formed by the intersections between all the $\mathcal{L}(Y^k)$'s and the surface of the $\frac{1}{8}$ sphere.

vides better mapping capability than the sigmoidal activation function. 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 PFUs 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 an MNN with self-recruitment.

A different analysis on the realization of eqn (9), based on interpreting eqn (9) in the PFU space instead of sample space, can be found in Appendix B.

2.2. Gaussian Potential Function Network

A Gaussian potential (activation) function, an unnormalized form of Gaussian density function, is selected for the construction of a Potential Function Network (PFN), since the function is highly nonlinear, provides good locality for incremental learning, and has many well-defined mathematical features. A Gaussian potential function ψ_i is defined by

$$\psi_i = \psi(\mathbf{x}, \mathbf{p}_i) = e^{-d(\mathbf{x},\mathbf{p}_i)/2} \tag{13}$$

$$d(\mathbf{x}, \mathbf{p}_i) = d(\mathbf{x}, \mathbf{m}^i, \mathbf{K}^i) = (\mathbf{x} - \mathbf{m}^i)^t\mathbf{K}^i(\mathbf{x} - \mathbf{m}^i) \tag{14}$$

where $\mathbf{x}$ represents an input pattern, $\mathbf{m}^i$ and $\mathbf{K}^i$ represent respectively the mean vector and the shape matrix (defined by the inverse of the covariance matrix) of the ith potential function.

$d(\mathbf{x}, \mathbf{m}^i, \mathbf{K}^i)$ can be rewritten as an expanded form:

$$d(\mathbf{x}, \mathbf{m}^i, \mathbf{K}^i) = \sum_j\sum_k k^i_{jk}(x_j - m^i_j)(x_k - m^i_k) \tag{15}$$

where x_j is the jth element of $\mathbf{x}$, m^i_j is the jth element of $\mathbf{m}^i$, and k^i_{jk} is the (j, k)th element of $\mathbf{K}^i$.

Without loss of generality, k^i_{jk} can be represented based on the marginal standard deviations σ^i_j and σ^i_k, and the correlation coefficient h^i_{jk}:

$$k^i_{jk} = \frac{h^i_{jk}}{\sigma^i_j\sigma^i_k} \tag{16}$$

where σ^i_j is positive real and $h^i_{jk} = 1$ *if* $j = k$ and $|h^i_{jk}| \le 1$ otherwise.[1]

[1] h^i_{jk} is assumed to satisfy the condition, $\Sigma_{k\neq j}|h^i_{jk}| \le 1\ \forall j$, for the positive semidefiniteness of shape matrix, $\mathbf{K}^i$.

Instead of using the general form of k^i_{jk} given by eqn (16), it is possible to use a simpler but a more restricted form of k^i_{jk} given by:

$$k^i_{jk} = \begin{cases} \frac{1}{\sigma_j^2} & \text{if } j = k \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

Equation (17) implies that the principle axes of the Gaussian potential should be aligned with the reference axes of the input space. However, the amount of flexibility loss due to the use of eqn (17) can be compensated by increasing the number of PFUs.

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 potential function units (GPFUs), which produce Gaussian potential functions. The weighted output values of the GPFUs are summed by the connection between the hidden layer and the output layer in order to synthesize the required potential fields. The three-layered PFN with the GPFUs configured at the hidden layer is called as the Gaussian Potential Function Network (GPFN).

Figure 2(a) illustrates the schematic diagram of a GPFN and Figure 2(b) shows a detailed structure of the ith GPFU. The calculation of eqn (15) starts with the subtraction of the mean vector of the ith GPFU from the input vector at the subtraction nodes. Then the components of the vector obtained at the subtraction nodes are cross-correlated among themselves (i.e., outer product of the two same vectors) by the cross-correlator to obtain N^2 cross-correlated terms. Each cross-correlated term is multiplied by the corresponding k^i_{jk} of the shape matrix $\mathbf{K}^i$ at the multiplication nodes and summed for d_i. The output of the GPFU is then generated by exponentiating d_i. Note that, for a GPFN producing multiple outputs, we opt for each output being generated independently by its own set of hidden units (GPFUs). This makes learning simpler.

3. HIERARCHICALLY SELF-ORGANIZING LEARNING

3.1. Motivation and Description

It may not be possible to train a neural network to reach a desired level of performance if the network does not have enough computational units, or the learning algorithm fails to find the optimal network parameters. It is thus quite attractive to develop a new type of learning algorithm capable of automatically recruiting new computational units whenever necessary for improving network performance. Here, we propose such type of learning algorithm called the Hierarchically Self-Organizing Learning (HSOL) algorithm.

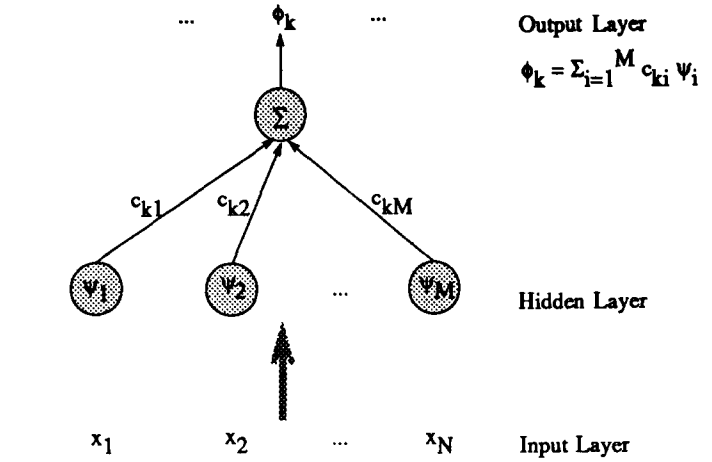
In the HSOL algorithm, a GPFU is associated with (a) the *accommodation boundary* defined in the input space and (b) the *class representation* defined in the output space. The accommodation boundary of a GPFU defines a region of input space upon which the corresponding GPFU can have an influence, and plays a role similar to the vigilance factor used in the ART2 (Carpenter & Grossberg, 1987), an automatic clustering algorithm based on adaptive resonance theory. If a new sample falls within the accommodation boundary of one of the currently existing GPFUs, which has the same class representation as that of the new sample, then, the network will not generate a new GPFU but accommodate the new sample by updating the parameters of existing GPFUs. Otherwise, the network will recruit a new GPFU. Most importantly, in HSOL, the accommodation boundaries of individual GPFUs are not fixed but *adjusted dynamically* in such a way as to achieve hierarchical learning: initially, the accommodation boundaries are set large for achieving rough but global learning, but gradually reduce to a smaller size for fine learning. Note that, as indicated previously, a Gaussian potential function serves better for implementing hierarchical learning due to its locality property. In general, the HSOL has the following implications on learning:

1. It starts with learning global mapping features based on a small number of computational units with larger size of accommodation boundaries and then proceeds to learning finer mapping details and increasing the number of computational units accordingly by reducing the size of accommodation boundaries.
2. It changes the dimension and shape of the error surface, that is, the surface of the error function defined in terms of the network parameters by increasing the number of computational units, when the performance of error convergence is considerably degraded. This helps to avoid the risk of sticking on a flat or a very mildly sloping surface which might cause trouble in backpropagation or steepest descent learning.

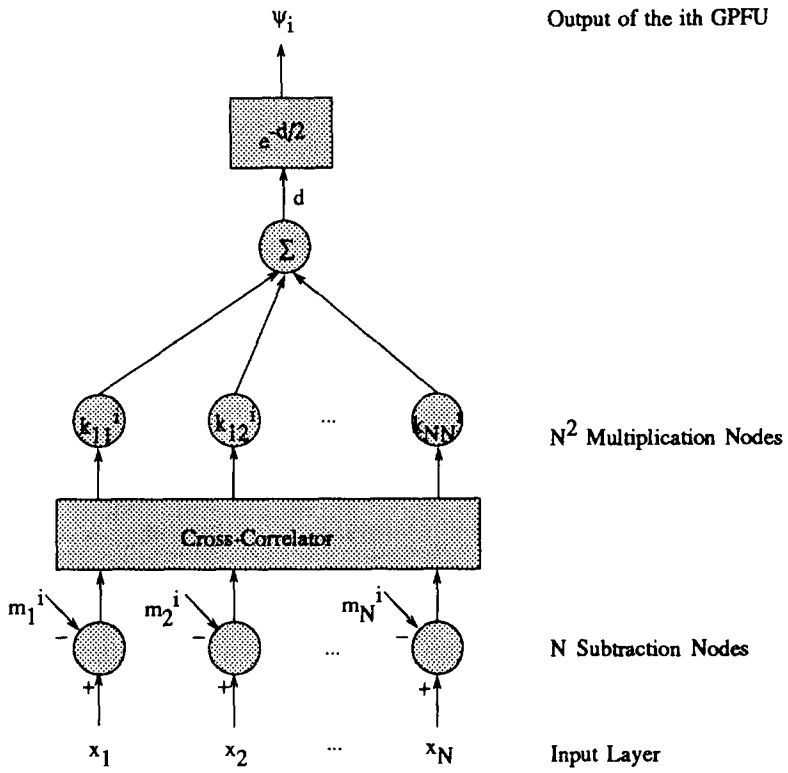
To describe an accommodation boundary, we introduce an *effective radius* or a *Mahalanobis distance* (Duda & Hart, 1973), r_i, of the ith GPFU in the form of a *hypersphere*, H_i, defined in the input space:

$$H_i(r_i) = \{\mathbf{x} | d(\mathbf{x}, \mathbf{m}^i, \mathbf{K}^i) \leq r_i^2\}. \quad (18)$$

A GPFU is assigned to its class representation, at



(a) The connection between the output layer and the hidden layer of GPFUs.



(b) The connection between the input layer and the ith GPFU.

FIGURE 2. The schematic diagram of a GPFN.

the time when it is recruited, by the same class as the class of teaching pattern that invoked its recruitment. Note that a set of classes can be predefined in the output space, which may be the pattern classes when the network is engaged in pattern classification, or the signs of the output (more generally a set of the subranges of the output space) when the network is engaged in function approximation.

The criteria for determining when to recruit new GPFU is summarized in the following Accommodation/Generation rules:

- If the input sample, $\mathbf{x}_p$ of a teaching pattern[2] is located within the hypersphere, $H_i(r_i)$, of the ith

[2]A teaching pattern, $(\mathbf{x}_p, \mathbf{t}_p)$, consists of a pair of input sample $\mathbf{x}_p$ and output sample, $\mathbf{t}_p$.

GPFU of the current network, which belongs to the same class representation as that of output sample, $\mathbf{t}_p$, then a new GPFU will not be invoked. In this case, the teaching pattern is accommodated by the ith GPFU, and the network simply updates and parameters.

- If the input sample, $\mathbf{x}_p$ of a teaching pattern, is not located within the hypersphere of any GPFU which has the same class representation as that of the output sample, $\mathbf{t}_p$, a new GPFU is generated at the position of the input sample, $\mathbf{x}_p$ of a teaching pattern.

To check whether a teaching pattern falls inside the hypersphere H_i of the ith GPFU, the output value of the ith GPFU for the given teaching pattern is compared with the following reference value of the ith GPFU, G_i, defined by its effective radius, r_i:

$$G_i = e^{-(r_i^2/2)}. \tag{19}$$

If the output of the ith GPFU is greater than G_i, it is considerd that a teaching pattern falls inside the hypersphere H_i of the ith GPFU.

As mentioned previously, the automatic adjustment of the accommodation boundary or the effective radius r_i provides the network with the capability of hierarchical learning. This can be carried out by either of the following two methods. The first method is based on reducing the effective radius of each GPFU gradually, starting from a large radius, according to the predetermined monotonously decreasing function. This method is simple and easily implementable but sensitive to the selected function in terms of the number of total GPFUs generated. For instance, if the effective radii of GPFUs are reduced too rapidly, the network generates more GPFUs than the minimum required because individual GPFUs may not have enough time to converge to their optimal shapes. On the other hand, if the effective radii of GPFUs are reduced too slowly, the network consumes a large number of learning cycles, although it eventually generates the minimum necessary number of GPFUs. Accordingly, the selection of a proper rate for the reduction of the effective radii is essential to successfully generate the minimum necessary number of GPFUs, and achieve a desirable learning speed.

The second method is based on reducing the radii of individual GPFUs according to the progress of learning. The reduction of the radii of individual GPFUs invokes the generation of more GPFUs, and so enables the network to learn the details. Therefore, the best time for a GPFU to reduce its radius is when the network performance to further learning becomes saturated with the currently available GPFUs. Figure 3 illustrates the procedure of HSOL by a flow chart, where the adjustment of accommodation boundary is based on the saturation of network performance.

Now, let us investigate a method of detecting the saturation of network performance. A simple way of measuring the progress of learning is by defining the *performance index*, P as follows:

$$P \equiv e^{-E_{rms}} \tag{20}$$

with E_{rms} representing the *root mean square error* for N teaching patterns. E_{rms} can be calculated by

$$E_{rms} \equiv \sqrt{\frac{2}{MN} \sum_{p=1}^{N} E_p} \text{ with} \tag{21}$$

$$E_p \equiv \frac{1}{2} \sum_{j=1}^{M} (t_{pj} - \phi_{pj}(\mathbf{n}_j))^2 \tag{22}$$

where M represents the number of output units, t_{pj} represents the jth element of the desired output vector defined by the pth teaching pattern, ϕ_{pj} represents the jth element of the actual output vector for the pth teaching pattern and $\mathbf{n}_j$ represents a column vector which is the collection of all parameters associated with the jth output unit.

To measure the saturation of network performance, we need to monitor the variation of P with respect to time or alternatively, we define the *parameter saturation vector*, $\mathbf{s}_j$ for the jth output unit based on the following difference equation:

$$\mathbf{s}_j(p) = \alpha \frac{\partial E_p}{\partial \mathbf{n}_j} + (1 - \alpha)\mathbf{s}_j(p - 1) \tag{23}$$

where α is a positive constant between 0 and 1, representing the decaying factor of $\partial E_p / \partial \mathbf{n}_j$, and p represents the pth teaching pattern presented to the network.

The purpose of defining $\mathbf{s}_j$ is to monitor $\partial E_p / \partial \mathbf{n}_j$ to see whether the network parameters have been sufficiently adjusted for E_p to reach its extremum with the currently available number of GPFUs. Note that eqn (23) results in the following solution of $\mathbf{s}_j$ at the mth presentation of teaching pattern with the assumption that $\mathbf{s}_j(0) = \mathbf{0}$:

$$\mathbf{s}_j(m) = \alpha \sum_{l=1}^{m} (1 - \alpha)^{m-l} \frac{\partial E_l}{\partial \mathbf{n}_j}(l - 1) \tag{24}$$

where l and m represent respectively the lth and the mth iteration ($m > l$). Based on eqn (24), eqn (23) can be interpreted as follows: (a) $\mathbf{s}_j$ provides the weighted average of $\partial E_p / \partial \mathbf{n}_j$ over the horizon of learning iterations, filtering out the high frequency components of $\partial E_p / \partial \mathbf{n}_j$, where the weights decays exponentially in the backward direction toward the initial iteration. (b) If, for a considerable number of iterations, the network parameters remain stationary at a point, $\partial E_p / \partial \mathbf{n}_j = \mathbf{0}$, or wander around a point

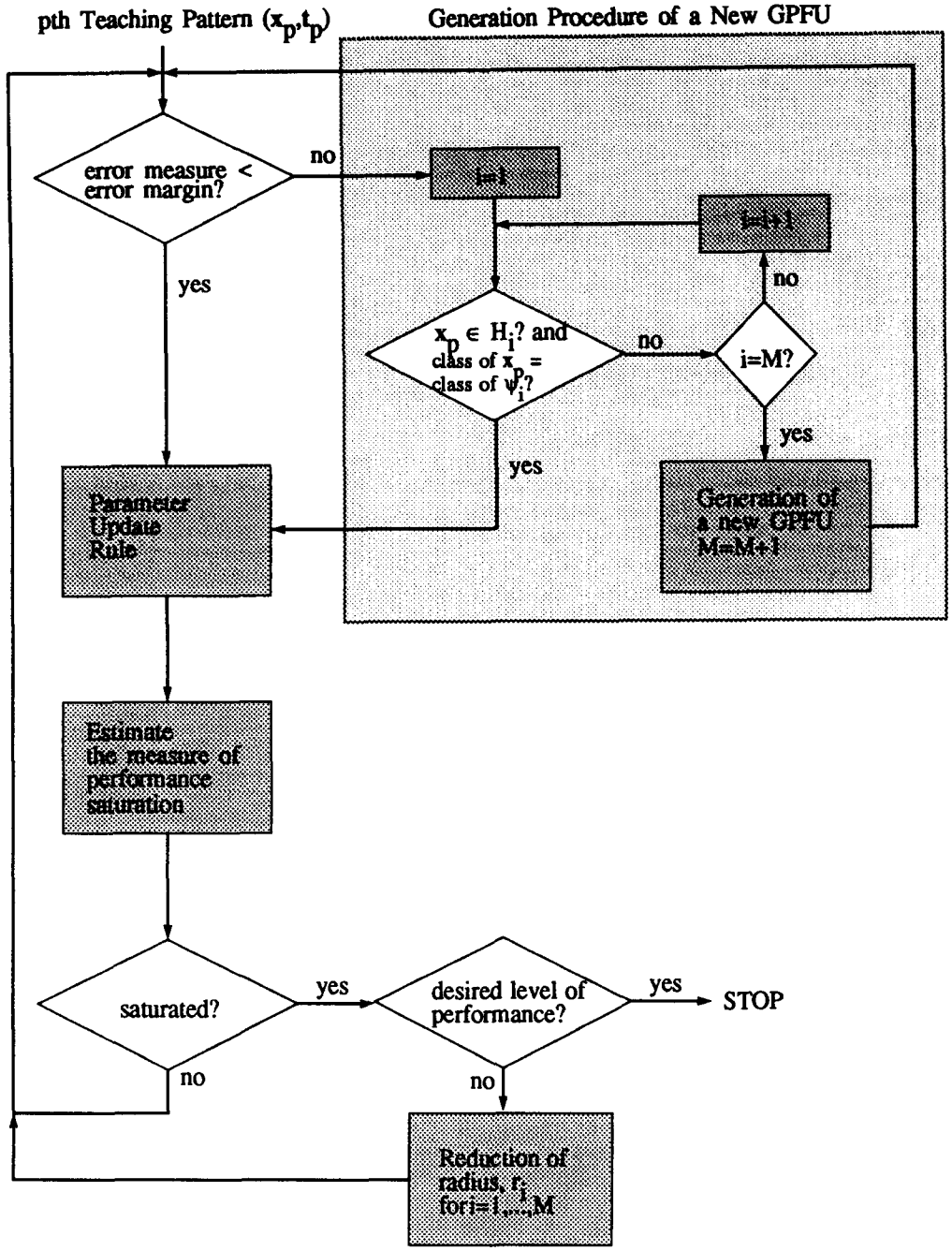


FIGURE 3. The procedure of Hierarchically Self-Organizing Learning in which the adjustment of accommodation boundary is based on the saturation of network performance: M, H_i and r_i represent respectively the number of GPFUs, effective radius and the hypersphere of the ith GPFU.

in the parameter space such that the average of $\partial E_p/\partial \mathbf{n}_j$ becomes close to zero, the magnitude of $\mathbf{s}_j$ gradually decreases toward zero. Therefore, by monitoring the value of $\|\mathbf{s}_j\|$, the saturation of the network performance to further learning can be detected.

The necessity of using the weighted average of $\partial E_p/\partial \mathbf{n}_j$, instead of directly using $\partial E_p/\partial \mathbf{n}_j$, comes from the fact that $\partial E_p/\partial \mathbf{n}_j$ not only varies with fluctuations but also generates spurious data frequently. In fact, $\|\mathbf{s}_j\|$ may not converge to zero but rather to a small value near zero, since $\partial E_p/\partial \mathbf{n}_j$ may drift near or around zero during saturation. Such a small saturated value of $\|\mathbf{s}_j\|$ is not known apriori and may vary according to the different sets and orders of teaching patterns. This implies that the use of a constant threshold for $\|\mathbf{s}_j\|$ for the detection of performance saturation may result in such a situation that it may need to go through an extremely long but unnecessary learning iterations to reach the termination condition, or the network may never be able to reach the termination condition at all. To resolve this problem, we adopt an adaptive saturation detec-

tion scheme which detects the saturation based on whether the integration of the inverse of $\|\mathbf{s}_j\|$ exceeds the vaue of $\|\mathbf{s}_j\|$. This scheme adaptively controls the termination threshold according to the convergence envelope of $\|\mathbf{s}_j\|$ and enforces the proper termination.

To be more precise, let us define ρ_j as the *saturation criteria* defined by the integration of the inverse of $\|\mathbf{s}_j\|$ for the jth output. ρ_j can be calculated based on the following difference equation:

$$\rho_j(p) = \begin{cases} \rho_j(p-1) + \beta \dfrac{\sqrt{d_j}}{\|\mathbf{s}_j(p)\|} & \text{if } p > p_0 \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

where d_j is the dimension of $\mathbf{s}_j$, β is a small positive constant representing the increment rate of ρ_j and p_0 is the delay factor which will be explained shortly. Then the decision whether the network performance is saturated or not, is made by comparing ρ_j with $\|\mathbf{s}_j\|/\sqrt{d_j}$. ρ_j is increased slowly (rapidly), should $\|\mathbf{s}_j\|$ converges to a larger (smaller) value, so as to provide a proper time for the network parameters to be saturated. Note that the integration should start only after a certain number of teaching patterns are presented, in order to avoid the situation that the small values of $\|\mathbf{s}_j\|$s during the initial periods of iterations due to the initial assignment of zero to $\|\mathbf{s}_j\|$ disturb the adaptive detection scheme. We let the integration start after $1/\alpha$ iterations (refer to eqn (23)), that is, $p_0 = [1/\alpha]$, since it is seen from eqn (24) that $[1/\alpha]$ iterations makes $\mathbf{s}_j$ reach approximately[3] 63% of $\partial E_p/\partial \mathbf{n}_j$, assuming small α and constant $\partial E_p/\partial \mathbf{n}_j$. The simulation results indicate that eqn (23) and eqn (25) provide a robust scheme for determining when to reduce the effective radii of individual GPFUs.

3.2. Learning Algorithm

The learning algorithm is composed of two parts: the first part is concerned with the adjustment of the network parameters and the second part is concerned with the recruitment of the minimum necessary number of GPFUs based on adjusting the accommodation boundaries of individual GPFUs.

3.2.1. *Parameter Update.* The network parameters are updated based on the Backpropagation learning algorithm (Rumelhart et al., 1986).

Parameter Update Rules. The parameter vector of the jth output unit, $\mathbf{n}_j$, $\mathbf{n}_j \equiv [\mathbf{w}_j^t, \mathbf{m}_j^t, \sigma_j^t, \mathbf{h}_j^t]^t$ is updated by

$$\mathbf{n}_j^{\text{new}} = \mathbf{n}_j^{\text{old}} + \eta \Delta \mathbf{n}_j \tag{26}$$

where η is the positive constant called the *learning rate*.

The directional vector, $\Delta \mathbf{n}_j = [\Delta \mathbf{w}_j^t, \Delta \mathbf{m}_j^t, \Delta \boldsymbol{\sigma}_j^t, \Delta \mathbf{h}_j^t]^t$, along which $\mathbf{n}_j$ should be updated, can be derived from the gradient descent of E_p defined in eqn (22), as listed in the following set of equations. Note that in the following equations, N and M respectively represent the dimensions of input and output vectors and the subscript p representing the pth teaching pattern is omitted from the equations for notational convenience.

- The weight between the jth output and the ith GPFU:

$$\Delta w_{ji} = -\frac{\partial E_p}{\partial w_{ji}} = (t_j - \phi_j)\psi_i. \tag{27}$$

- The jth element of the mean vector, $\mathbf{m}^i$:

$$\Delta m_j^i = -\frac{\partial E_p}{\partial m_j^i} = \sum_{l=1}^{N} k_{jl}^i (x_l - m_j^i)\psi_i \sum_{k=1}^{M} (t_k - \phi_k) w_{ki}. \tag{28}$$

- The marginal standard deviation, σ_j^i:

$$\Delta \sigma_j^i = -\frac{\partial E_p}{\partial \sigma_j^i} = \sum_{l=1}^{N} k_{jl}^i \frac{(x_j - m_j^i)(x_l - m_l^i)}{\sigma_j^i} \psi_i \sum_{k=1}^{M} (t_k - \phi_k) w_{ki}. \tag{29}$$

- The correlation coefficient, h_{jk}^i:

$$\Delta h_{jk}^i = -\frac{\partial E_p}{\partial h_{jk}^i} = -\frac{1}{2} \frac{(x_j - m_j^i)(x_k - m_k^i)}{\sigma_j^i \sigma_k^i} \psi_i \sum_{k=1}^{M} (t_k - \phi_k) w_{ki}. \tag{30}$$

For more details on the derivation of the above equations, see Appendix C.

3.2.2. *HSOL Algorithm.* Initially, there is no GPFU assigned to the network, and the output of the network is set to zero. The following HSOL algorithm is then applied to the network to automatically create and shape GPFUs and adjust weight vectors for the kth output.

Hierarchically Self-Organizing Learning Algorithm. Step 1. Initialization:

- Set $i = 1$, $j = 0$ and $p = 0$, where i represents the number of learning cycles, j represents the number of GPFUs, and p represents the number of patterns presented to the network.
- Set $s_k = \mathbf{0}$ and $\rho_k = 0$ (refer to eqn (23) and eqn (25)).

Step 2. Invoke the ith learning cycle, where one-learning cycle implies the random presentation of all

[3] From eqn (24), let $c = \sum_{i=1}^{[1/\alpha]} \alpha(1-\alpha)^{i-1} = 1 - (1-\alpha)^{[1/\alpha]}$. Then,

$$\lim_{\alpha \to 0} c = 1 - \lim_{\alpha \to 0} (1-\alpha)^{[1/\alpha]} = 1 - e^{\lim_{\alpha \to 0} [1/\alpha] \ln(1-\alpha)} = 1 - e^{-1} \approx 0.63$$

the teaching patterns in the pool to the network. The procedure of one learning cycle is as follows:

Step 2.1. Get the next teaching pattern.

Step 2.2. Set $p = p + 1$

Step 2.3. Apply the following kernel procedure:

C1. If $|t_{pk} - \phi_{pk}| > \varepsilon_m$, where t_{pk} and ϕ_{pk} respectively represent the desired and actual values of the kth output unit for the pth teaching pattern, and ε_m represents the error margin, then do the following:

- If there is a GPFU having the same class representation as that of the pth teaching pattern and the pth teaching pattern falls inside the hypersphere of the GPFU, apply the parameter update rules.
- If there is no such GPFU, then generate a new GPFU:
  —Set $j = j + 1$ and $p = 0$.
  —Set $\mathbf{s}_k = \mathbf{0}$ and $\rho_k = 0$.
  —The following parameter values are assigned to the new GPFU:
  *The Mean Vector, $\mathbf{m}^j$ = the input sample, $\mathbf{x}_p$ of the pth teaching pattern.
  *The Weight Value, c_j = output sample, $\mathbf{t}_p$ of the pth teaching pattern.
  *The Shape Matrix, $\mathbf{K}^j = 1/\sigma_0^2\mathbf{I}$, where σ_0 is the predefined nominal variance.
  *The effective radius, $r_j = r_0$, where r_0 is the predefined initial effective radius.
  —Go to Step 2.1.

C2. If $|t_{pk} - \phi_{pk}| \leq \varepsilon_m$, then apply the parameter update rules.

C3. Calculate $\|\mathbf{s}_k\|/\sqrt{d_k}$ based on eqn (23).

C4. Calculate ρ_k based on eqn (25).

C5. If $\|\mathbf{s}_k\|/\sqrt{d_k} < \rho_k$, then reduce the radius the effective radius of individual GPFUs:

$$r_l^{\text{new}} = \begin{cases} r_l^{\text{old}} * r_d & \text{for } l = 1, \cdots, j, \quad \text{if } r_l > r_L \\ r_l^{\text{old}} & \text{for } l = 1, \cdots, j, \quad \text{otherwise} \end{cases}$$

where r_L is the lower bound of radius and r_d is the radius decrement rate.

Step 2.4. If all the teaching patterns are presented, go to the next step. Otherwise, go to Step 2.1.

Step 3. Set $i = i + 1$

Step 4. If the network shows satisfactory performance, then stop. If not, go to Step 2.

4. SIMULATION

Simulations were conducted for the six sets of teaching patterns shown in Figure 4. The first four sets of teaching patterns, (a), (b), (c) and (d) represent binary functions of two output classes $+1$ and -1, whereas the last two sets of teaching patterns, (e) and (f) represent continuous functions, $\sin(\pi x_1)\cos(0.5\pi x_2)$ and $\cos(4\pi x_1)\cos(4\pi x_2)e^{-10(x_1^2 + x_2^2)}$ respectively, defined over 2 dimensional input space. At each learning cycle, all the teaching patterns of a set were presented to the network in a random order. The teaching patterns of the last two sets were generated respectively from the given continuous functions, such that at each learning cycle, 10 teaching patterns are randomly selected from the corresponding function and presented to the network.

It was considered that the network reached a satisfactory level of performance through training, when the following condition was met: (a) the absolute error $|t_{pk} - \phi_{pk}|$ is less than the predetermined error margin, ε_m for every teaching patterns for the first four sets of teaching patterns, (a), (b), (c) and (d), and (b) the rms error E_{rms} defined by eqn (22) is less than ε_m for the last two sets of teaching patterns, (e) and (f).

The learning rate was chosen carefully because (a) an excessive learning rate can cause the algorithm to fluctuate and eventually diverge, and (b) a small learning rate can cause slow convergence, although no fluctuation occurs.

The error margin ε_m was set according to the nature of the desired output. For example, if the desired output is represented by binary value, the error margin need not be set to a very small value, because it is satisfactory as long as the signs of the desired and the actual output values agree. In this case, the desired output can be realized by thresholding the actual output after learning is completed. If the desired output has a continuous value, the error margin ε_m should be set to a value small enough for the network to map the given function accurately. However, the decrease of ε_m causes the increase of the number of GPFUs to be generated. Therefore, the selection of ε_m should consider the trade-off between accuracy and complexity in network realization.

The initial assignment of the marginal standard deviation σ_0 also affects the performance of learning. If σ_0 is set too high, it is difficult to train the network to accurately represent those teaching patterns densely distributed in the input space. If σ_0 is set too low, it takes too long to train the network to accurately represent the teaching patterns coarsely distributed in the input space. Therefore, it is recommended that a proper value of σ_0 be set according to the distribution of teaching patterns in the input space. This can be done based on the preanalysis of input samples in terms of local density distribution or minimum distance between input samples.

The initial effective radius of a GPFU, r_0, was chosen large enough to cover most of the input space or almost all the input samples, so that initially the GPFN could carry out learning with a small number of GPFUs. The lower bound of the effective radius,

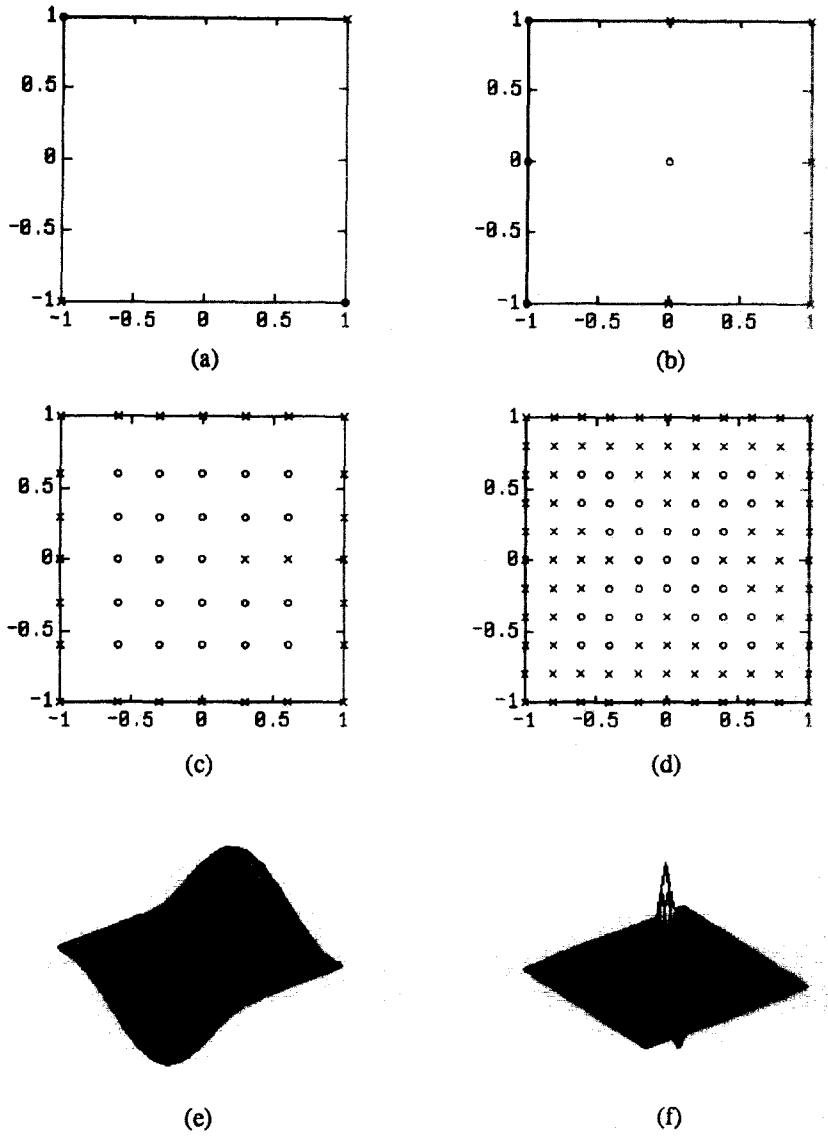


FIGURE 4. (a), (b), (c), (d), (e) and (f) represent the six sets of teaching patterns used for training the GPFN: The teaching patterns (a), (b), (c) and (d) represent binary functions of two output classes, +1 and −1, (which are denoted respectively by o and x) defined over the two-dimensional sample points. The teaching patterns, (e) and (f) represent continuous functions, $\sin(\pi x_1)\cos(0.5\pi x_2)$ and $\cos(4\pi x_1)\cos(4\pi x_2)e^{-10(x_1^2+x_2^2)}$ respectively, defined over two-dimensional input space.

r_L, was used to curtail the number of GPFUs generated for a continuous mapping. Setting r_L low allows the accurate realization of a continuous function, but only at the expense of increased number of GPFUs.

The recruitment of new GPFUs was controlled by the three parameters: r_d, α and β. r_d, the radius decrement rate, was chosen small enough to reduce the radii of GPFUs slowly, so that the most troublesome positions, which have large errors between the desired and the actual outputs but are not properly covered by existing GPFUs, can be found and cov-

TABLE 1
Actual Parameter Values Assigned for Each Set of Teaching Patterns

Teaching Pattern Set	(a)	(b)	(c)	(d)	(e)	(f)
Learning Rate, η	0.02	0.02	0.002	0.002	0.002	0.002
Error Margin, ε_m	1	1	1	1	0.1	0.1
Initial Standard Deviation, σ_0	2	2	0.2	0.2	0.1	0.1
Initial Effective Radius, r_0	3	3	30	30	60	60
Lower Bound of Radius, r_L	0	0	0	0	2	2
Radius Decrement Rate, r_d	0.9986	0.9986	0.9986	0.9986	0.9986	0.9986
Decaying Factor, α	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
Increment Rate, β	10^{-10}	10^{-10}	10^{-10}	10^{-10}	10^{-10}	10^{-10}

TABLE 2
A Summary of Simulation Results

Teaching Pattern Set	(a)	(b)	(c)	(d)	(e)	(f)
The No. of Learning Cycles	400	1500	4500	4500	5000	10000
The Estimated No. of Minimum GPFUs	2	2	4	6	NA	NA
The No. of GPFUs Generated	2	2	5	8	2	14
E_{rms}[a] after Learning	0.0001	0.2866	0.3634	0.2916	0.1048	0.0458

[a] The E_{rms}s of the teaching pattern set e and f, are calculated based on 1000 randomly generated patterns from the given function.

ered first. α, the decaying factor for $\partial E_p/\partial \mathbf{n}_k$, affects $\mathbf{s}_k$ as follows: If α is small, $\|\mathbf{s}_k\|/\sqrt{n_k}$ has small fluctuations in spite of a large fluctuations of $\partial E_p/\partial \mathbf{n}_k$; on the other hand, if α is large, $\|\mathbf{s}_k\|/\sqrt{n_k}$ has large fluctuations following the fluctuations of $\partial E_p/\partial \mathbf{n}_k$'s. For the detection of the saturation of network performance, α was set small enough to have small fluctuations in $\|\mathbf{s}_k\|/\sqrt{n_k}$ but large enough to follow the current trend of the variations of $\partial E_p/\partial \mathbf{n}_k$. In practice, α can be chosen inversely proportional to the number of teaching patterns. β, the increment rate of ρ_k, affects the detection of network saturation as follows: If β is large, ρ_k increases rapidly and the network prematurely determines the occurrence of saturation; on the other hand, if β is small, the network determines the occurrence of saturation

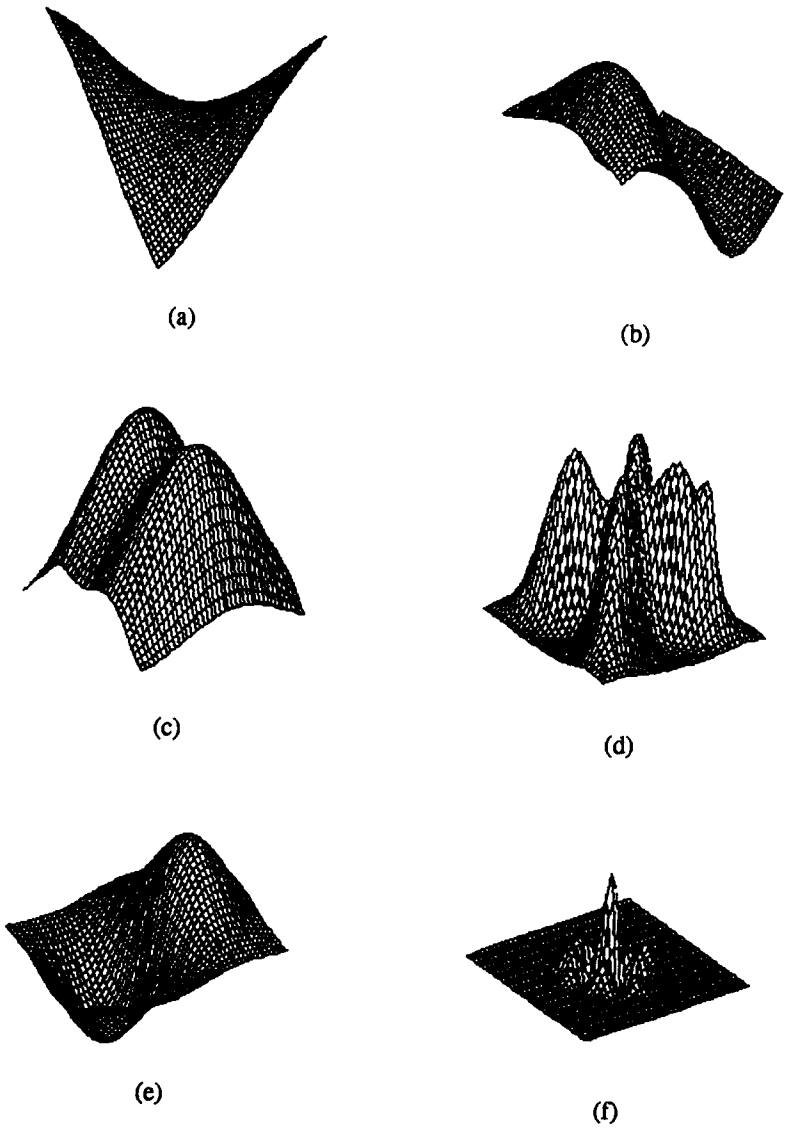


FIGURE 5. 3D representation of potential fields synthesized over 2D input space as a result of learning: The potential fields, (a), (b), (c), (d), (e), and (f) shown in this figure correspond respectively to the individual teaching patterns, (a), (b), (c), (d), (e) and (f) of Figure 4.

a long time after the actual saturation occurred. Therefore, the selection of β should consider the above trade-off.

Table 1 illustrates the actual parameter values used in the simulations for the six sets of teaching patterns. Table 2 presents a brief summary of simulation results, including the number of learning cycles required and the number of GPFUs generated, for individual sets of teaching patterns. Figure 5 illustrates the 3D representation of the potential fields synthesized over the 2D input space as a result of learning. The potential fields, (a), (b), (c), (d), (e) and (f) of Figure 5 corresponds respectively to the individual sets of learning patterns, (a), (b), (c), (d), (e) and (f) of Figure 4. Figure 6 shows decision boundaries obtained by thresholding the potential fields, (a), (b), (c) and (d) of Figure 5 with the zero threshold. Figure 7 shows the learning curves for the six sets of teaching patterns where the variations of curves represent e_{rms} and the number of GPFUs with respect to the number of learning cycles. The simulation results indicate the following: (i) The decision boundaries for the first four sets of teaching patterns, (a), (b), (c) and (d) with binary output values provide the perfect binary classification with optimal or near optimal number of GPFUs. (ii) A good approximation (around 5% error or less) of continuous functions were obtained with a small number of GPFUs for the last two sets of teaching patterns (e) and (f).

Finally, to show the effectiveness of a GPFN based on HSOL for pattern classification, a comparison is made for the first four sets of teaching patterns, between the GPFNs generated by the HSOL algorithm and the Backpropagation networks optimized by trial-and-error in terms of the number of hidden units (using sigmoidal activation functions). Table 3 describes the result of such a comparison: For the teaching pattern sets, (a) and (b), the GPFNs have the total number of parameters larger than that of the optimal Backpropagation networks. However, for the teaching pattern sets, (c) and (d), the GPFNs have the total number of parameters less than those of the optimal Backpropagation networks. Note that, even for the teaching pattern sets, (a) and (b), in order to obtain the decision boundaries similar to what the GPFNs could provide, the optimal Backpropagation networks re-

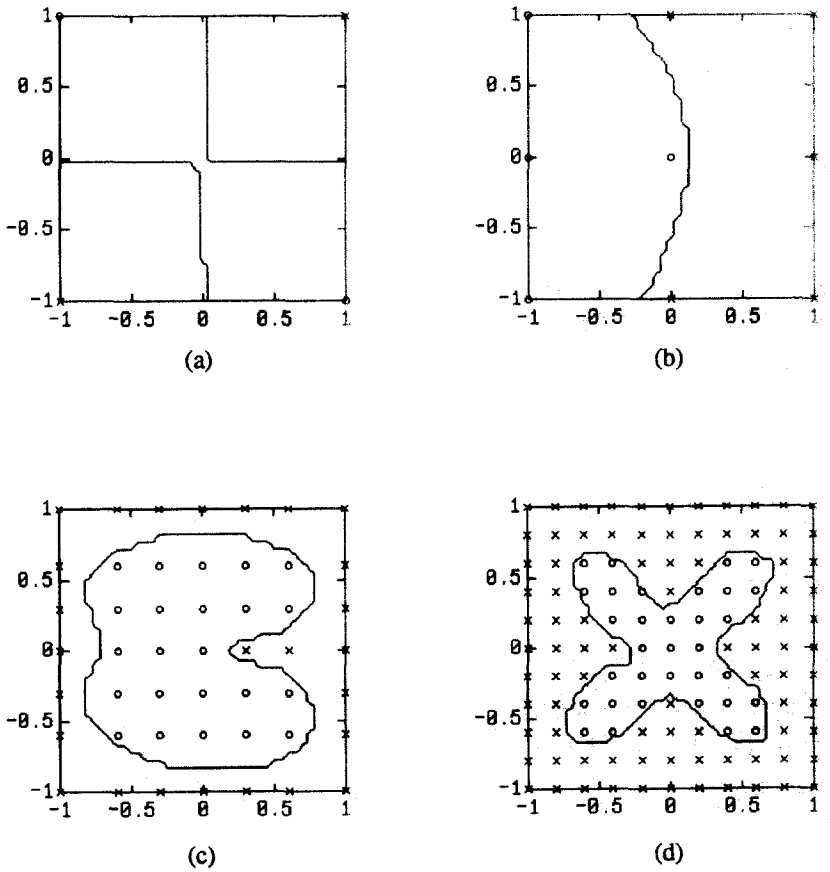


FIGURE 6. The decision boundaries obtained from the synthesized potential fields: The decision boundaries of the first four sets of teaching patterns are obtained respectively by thresholding the corresponding potential fields, (a), (b), (c) and (d) of Figure 5 with the zero threshold value.

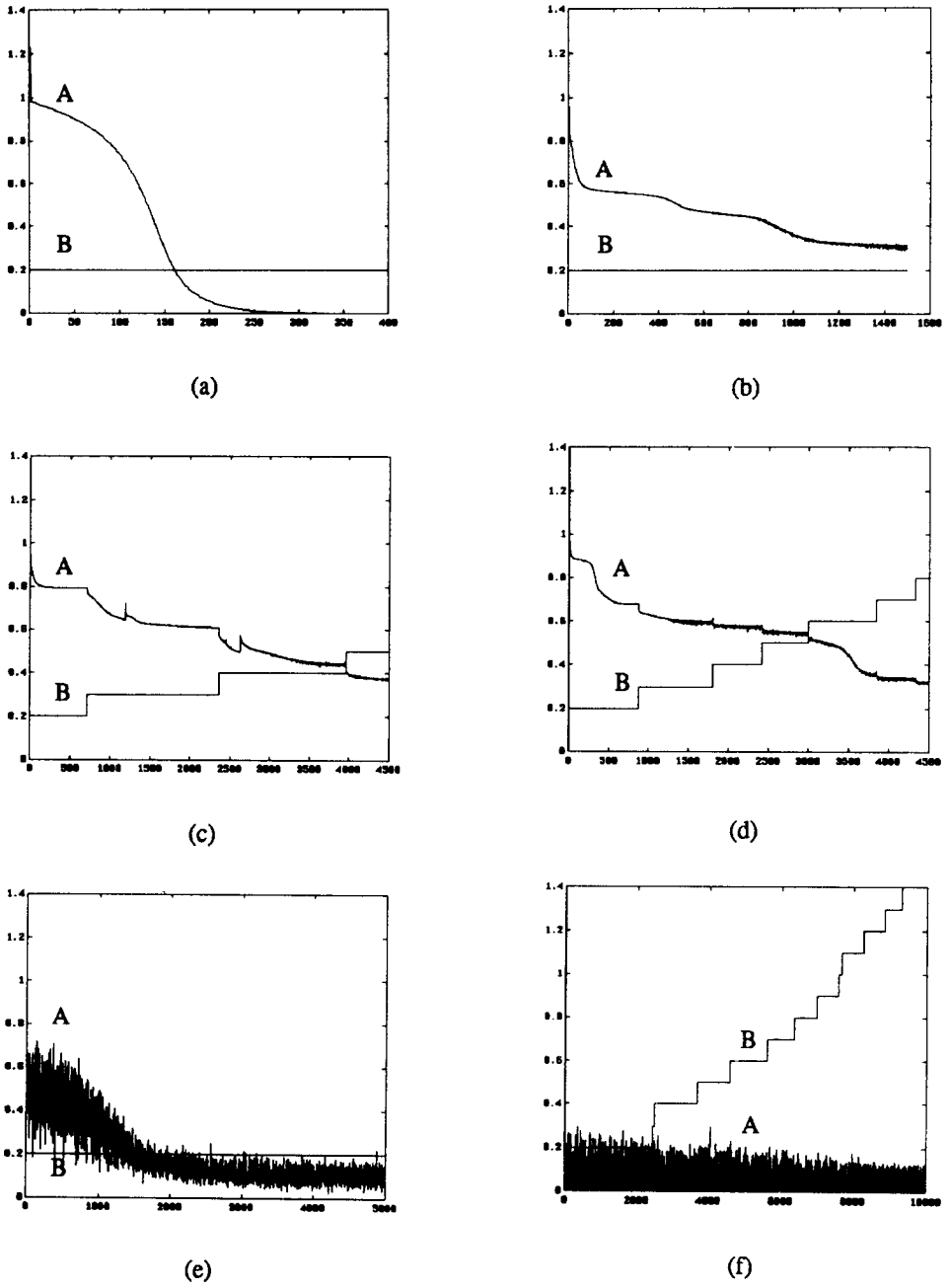


FIGURE 7. The learning curve A represents the variation of E_{rms} with respect to the number of learning cycles, whereas the learning curve B represents the increase of GPFUs along learning cycles, for the six sets of teaching patterns: E_{rms} is calculated based on the entire teaching patterns presented at each learning cycle. Note: The scales associated with y axis represents the actual value of E_{rms}. To obtain the number of GPFUs, multiply the scales shown in the figure by 10.

TABLE 3
Comparison between the GPFN Generated by HSOL and the Optimal Backpropagation Network

Teaching Pattern Set		(a)	(b)	(c)	(d)
GPFN	The No. of GPFUs Generated	2	2	5	8
	The No. of Total Parameters	12	12	30	48
Optimal Backpropagation Network	The estimated No. of Minimum Sigmoidal Functions[a]	2	2	9	13
	The No. of Total Parameters	8	8	33	50

[a] For teaching pattern set 3 and 4, two layers of hidden units are used.

quire more parameters than those of the GPFNs. This illustrates the power of a GPFN based on the HSOL algorithm.

5. CONCLUSION

This article has presented the following:

1. The design principle of a MNN using a nonsigmoidal activation function, such as a Gaussian function.
2. The HSOL algorithm which explores the incremental recruitment of hidden units based on the hierarchical learning of teaching patterns which is achieved by the control of the accommodation boundaries of individual hidden units.

The design principle developed for a MNN based on a nonsigmoidal activation function contributes to the advancement of a new methodology for designing a more general form of a MNN with powerful mapping capability. The presented HSOL algorithm is applicable to any MNN by properly defining its own accommodation boundaries. The HSOL algorithm contributes to the development of a new learning methodology based on self-organization and hierarchical learning, which may provide a solution to the problems encountered in conventional learning techniques due to the existence of local minima, flat surface error curvature, as well as structural inflexibility.

REFERENCES

Ackley, D., Hinton, G., & Sejnowski T. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, **9**, 147–169.

Aizerman, M., Braverman, E., & Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automatika i Telemekhanika*, **25**, 917–936.

Carpenter, G., & Grossberg, S. (1987). Art2: stable self-organization of pattern recognition codes for analog input patterns. *Applied Optics*, **26**, 4919–4930.

Duda, R., & Hart, P. (1973). *Pattern Classification and Scene Analysis*. New York: Wiley.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, **2**, 183–192.

Fukushima, K. (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, **36**, 193–202.

Fukushima, K. (1988). A neural network for visual pattern recognition. *IEEE Computer Magazine*, March, 65–75.

Hecht-Nielsen, R. (1987a). Counterpropagation networks. *Applied Optics*, **26**, 4979–4984.

Hecht-Nielsen, R. (1987b). Kolmogorov mapping neural network existence theorem. *IEEE International Conference on Neural Networks*, **3**, 11–13.

Hecht-Nielsen, R. (1989). Theory of the backpropagation neural network. *IEEE International Joint Conference on Neural Networks*, **1**, 593–605.

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational properties. *Proceedings of National Academy of Sciences (U.S.A.)*, **79**, 2554–2558.

Hopfield, J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of National Academy of Sciences (U.S.A.)*, **81**, 3088–3092.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, 359–366.

Irie, B., & Miyake, S. (1988). Capabilities of three-layered perceptrons. *IEEE International Conference on Neural Networks*, **1**, 641–648.

Kohonen, T. (1984). *Self-Organization and Associative Memory*. New York: Springer-Verlag.

Kolmogorov, A. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, **144**, 679–681.

Kosko, B. (1987). Adaptive bidirectional associative memories. *Applied Optics*, **26**, 4947–4960.

Koustougeras, C., & Papachristou, C. (1988). Training of a neural network model for pattern classification based on an entropy measure. *IEEE International Conference on Neural Networks*, **4**, 573–582.

Lippmann, R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, **4**, 4–22.

Lee, S., & Kil, R. M. (1988). Multilayer feedforward potential function network. *IEEE International Conference on Neural Networks*, **1**, 161–171.

Minsky, M., & Papert, S. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Parzen, E. (1962). On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, **33**, 1065–1076.

Reilly, D. L., Cooper, L. N., & Elbaum, C. (1982). A neural model for category learning. *Biological Cybernetics*, **45**, 35–41.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representations by error propagation. In D. Rumelhart, & J. McClelland (Eds.), *Parallel Distributed Processing: Exploration in the Microstructure of cognition*. Cambridge, MA: MIT Press.

Sprecher, D. (1965). On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, **115**, 340–355.

APPENDIX A

Kolmogorov/Sprecher Theorem

For each integer $n \geq 2$, there exists a real monotonic increasing function $\psi(x)$, $\psi([0, 1]) = [0, 1]$, dependent on n and having the following property:

For each preassigned number $\delta > 0$, there is a rational number $\varepsilon, 0 < \varepsilon \leq \delta$, such that every real continuous function of n variables, $\phi(\mathbf{x})$, defined on I^n, can be exactly represented by

$$\phi(\mathbf{x}) = \sum_{j=1}^{2n+1} \chi\left[\sum_{i=1}^{n} \lambda^i \psi(x_i + \varepsilon(j-1)) + j - 1\right]$$

where χ is a real and continuous function dependent upon ϕ, and λ^i is a constant independent of ϕ.

APPENDIX B

The Realizability of Mapping Functions

From eqn (9),

$$\begin{aligned}\mathbf{z} &= \mathbf{Yc} \\ &= [\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_N]^t\mathbf{c} \\ &= [\mathbf{r}_1^t\mathbf{c}, \mathbf{r}_2^t\mathbf{c}, \cdots \mathbf{r}_N^t\mathbf{c}]^t\end{aligned}$$

where $\mathbf{z} = [z_1, z_2, \cdots, z_N]^t$ and $\mathbf{r}_i$ is the ith row of $\mathbf{Y}$, $\mathbf{r}_i \equiv [\psi(\mathbf{x}_i, \mathbf{p}_1), \psi(\mathbf{x}_i, \mathbf{p}_2), \cdots, \psi(\mathbf{x}_i, \mathbf{p}_M)]^t$ *for* $i = 1, \cdots, N$. Note that $\mathbf{r}_i$, $i = 1, \cdots, N$ are $M \times 1$ vectors represented in the M dimensional PFU space.

The above equation can be rewritten as

$$[\mathbf{r}_1^t\mathbf{c} - z_1, \mathbf{r}_2^t\mathbf{c} - z2, \cdots, \mathbf{r}_N^t\mathbf{c} - z_N]^t = \mathbf{0}.$$

Therefore,

$$\mathbf{R}'\mathbf{c}' \equiv [\mathbf{r}_1', \mathbf{r}_2', \cdots, \mathbf{r}_N']^t\mathbf{c}' = \mathbf{0}.$$

where $\mathbf{r}_i' \equiv [\mathbf{r}_i^t : -z_i]^t : (M + 1) \times 1$ *vector*, $\mathbf{c}' \equiv [\mathbf{c}^t : 1]^t : (M + 1) \times 1$ *vector* and $\mathbf{R}' = [\mathbf{r}_1', \mathbf{r}_2', \cdots, \mathbf{r}_N']^t : N \times (M + 1)$ *matrix*.

To provide a solution for $\mathbf{c}'$, $\mathbf{r}_i'$, $i = 1, \cdots, N$ should reside in the manifold of less than or equal to M dimension, so that $\mathbf{c}'$ can be perpendicular to all $\mathbf{r}_i'$s. This can be tested by measuring the quantity Λ_p,

$$\Lambda_p \equiv \min_i \lambda_i$$

where λ_i, $i = 1 \cdots, M + 1$ are the eigenvalues of the cross-correlation matrix, $\mathbf{\Sigma}$, defined by

$$\mathbf{\Sigma} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{r}_i'\mathbf{r}_i'^t.$$

Then, the following theorem holds:

Theorem 2: $\mathbf{z} = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \cdots, \phi(\mathbf{x}_N)]^t$ *is exactly realizable if* $\Lambda_p = 0$ *and there exists an eigenvector with zero eigenvalue which is not* $[\mathbf{0}, 1]^t$.

Proof: $\Lambda_p = 0$ implies that $\mathbf{r}_i'$, $i = 1, \cdots, N$ are in the manifold of M or less than M dimensions. The latter condition is to guarantee that a vector, $\mathbf{c}'$, perpendicular to the manifold formed by $\mathbf{r}_i'$, $i = 1, \cdots, N$, is nontrivial, that is, $\mathbf{c}' \neq [\mathbf{0}:1]^t$. Q.E.D.

Theorem 2 implies that Λ_p can also be used as a criterion for the adjustment of the shape of PFUs. Note that, in Theorem 2, the eigenvectors with zero eigenvalues can be the solutions of $\mathbf{c}'$:

$$\mathbf{c}'^t\mathbf{\Sigma}\mathbf{c}' = 0 \Rightarrow \frac{1}{N}\sum_{i=1}^{N} (\mathbf{r}_i'^t\mathbf{c}')^2 = 0 \Rightarrow \mathbf{r}_i'^t\mathbf{c}' = 0 \ \forall i.$$

APPENDIX C

The Derivation of Parameter Update Rules

The network parameter update rules are derived here by taking the negative gradient of the error function, eqn (22). For convenience, the subscript p which represents the pth pattern is omitted in the following derivations.

1. Δw_{ji}

$$\Delta w_{ji} = -\frac{\partial E}{\partial w_{ji}},$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial \phi_j}\frac{\partial \phi_j}{\partial w_{ji}},$$

$$\frac{\partial E}{\partial \phi_j} = -(t_j - \phi_j), \text{ and}$$

$$\frac{\partial \phi_j}{\partial w_{ji}} = \psi_i.$$

Therefore,

$$\Delta w_{ji} = (t_j - \phi_j)\psi_i.$$

2. Δm_j^i

$$\Delta m_j^i = -\frac{\partial E}{\partial m_j^i},$$

$$\frac{\partial E}{\partial m_j^i} = \frac{\partial E}{\partial \psi_i}\frac{\partial \psi_i}{\partial d_i}\frac{\partial d_i}{\partial m_j^i},$$

$$\frac{\partial E}{\partial \psi_i} = \Sigma_k \frac{\partial E}{\partial \phi_k}\frac{\partial \phi_k}{\partial \psi_k} = -\Sigma_k(t_k - \phi_k)w_{ki},$$

$$\frac{\partial \psi_i}{\partial d_i} = -\frac{1}{2}\psi_i \text{ and}$$

$$\frac{\partial d_i}{\partial m_j^i} = -2\Sigma_l k_{jl}^i(x_l - m_j^i).$$

Therefore,

$$\Delta m_j^i = \Sigma_l k_{jl}^i(x_l - m_j^i)\psi_i\Sigma_k(t_k - \phi_k)w_{ki}.$$

3. Δk_{jk}^i

We have two types of parameters for k_{jk}^i. That is, σ_j^i and h_{jk}^i. Here, $\Delta\sigma_j^i$ and Δh_{jk}^i are derived separately.

- $\Delta\sigma_j^i$ for $j = k$

$$\Delta\sigma_j^i = -\frac{\partial E}{\partial \sigma_j^i} \text{ and}$$

$$\frac{\partial E}{\partial \sigma_j^i} = \frac{\partial E}{\partial \psi_i}\frac{\partial \psi_i}{\partial d_i}\frac{\partial d_i}{\partial \sigma_j^i}.$$

By the same derivation as Δm_j^i,

$$\frac{\partial E}{\partial \psi_i}\frac{\partial \psi_i}{\partial d_i} = \frac{1}{2}\psi_i\Sigma_k(t_k - \phi_k)w_{ki} \text{ and}$$

$$\frac{\partial d_i}{\partial \sigma_j^i} = -2\Sigma_l h_{jl}^i \frac{(x_l - m_j^i)(x_l - m_j^i)}{(\sigma_j^i)^2\sigma_j^i}.$$

Therefore,

$$\Delta\sigma_j^i = \Sigma_l k_{jl}^i \frac{(x_l - m_j^i)(x_l - m_j^i)}{\sigma_j^i}\psi_i\Sigma_k(t_k - \phi_k)w_{ki}.$$

- Δh_{jk}^i for $j \neq k$

$$\Delta h_{jk}^i = -\frac{\partial E}{\partial h_{jk}^i} \text{ and}$$

$$\frac{\partial E}{\partial h_{jk}^i} = \frac{\partial E}{\partial \psi_i}\frac{\partial \psi_i}{\partial d_i}\frac{\partial d_i}{\partial h_{jk}^i}.$$

By the same derivation as Δm_j^i,

$$\frac{\partial E}{\partial \psi_i}\frac{\partial \psi_i}{\partial d_i} = \frac{1}{2}\psi_i\Sigma_k(t_k - \phi_k)w_{ki} \text{ and}$$

$$\frac{\partial d_i}{\partial h_{jk}^i} = \frac{(x_j - m_j^i)(x_k - m_k^i)}{\sigma_j^i\sigma_k^i}.$$

Therefore,

$$\Delta h_{jk}^i = -\frac{1}{2}\frac{(x_j - m_j^i)(x_k - m_k^i)}{\sigma_j^i\sigma_k^i}\psi_i\Sigma_k(t_k - \phi_k)w_{ki}.$$

NOMENCLATURE

$\mathbf{c}$	Weight vector		ith shape matrix
$\mathbf{c}'$	Augmented column vector defined by $[\mathbf{c}^t : 1]^t$	$\mathbf{m}^i$	Mean vector of the ith GPFU
$\mathbf{c}^*$	Optimal solution of $\mathbf{c}$	m_j^i	The jth element value of the ith mean vector $\mathbf{m}^i$
c_i	Summation weight of the ith potential function	n_i	Dimension of $\mathbf{s}_j$
		$\mathbf{n}_j$	Parameter vector for the jth output unit
d	Weighted distance of GPFU	$\mathbf{p}_i$	Parameter vector of the ith potential function
d_i	Weighted distance of the ith GPFU	r_0	Initial effective radius of GPFU
h_{jk}^i	The (j, k)th correlation coefficient of the ith GPFU	r_i	Effective radius of ith GPFU
k_{jk}^i	The (j, k)th element of the	$\mathbf{r}_i$	Column vector

defined by $[\psi(\mathbf{x}_i, \mathbf{p}_1), \psi(\mathbf{x}_i, \mathbf{p}_2), \cdots, \psi(\mathbf{x}_i, \mathbf{p}_M)]^t$

$\mathbf{r}_i'$ Augmented column vector defined by $[\mathbf{r}_i': -z_i]^t$

r_d Radius decrement rate

r_L Lower bound of radius

$\mathbf{s}_j$ Parameter saturation vector for the jth output unit

t_{pj} The jth desired output value of the pth teaching pattern

w_{ji} Summation weight between the jth output unit and the ith GPFU

$\mathbf{x}$ Input pattern

x_i The ith element value of $\mathbf{x}$

$\mathbf{y}_i$ Column vector defined by $[\psi(\mathbf{x}_1, \mathbf{p}_i), \psi(\mathbf{x}_2, \mathbf{p}_i), \cdots, \psi(\mathbf{x}_N, \mathbf{p}_i)]^t$

$\mathbf{z}$ Column vector defined by $[\psi(\mathbf{x}_1), \psi(\mathbf{x}_2), \cdots, \psi(\mathbf{x}_N)]^t$

z_i the ith element of $\mathbf{z}$

E Error function

E_p Error function for the pth teaching pattern

E_{rms} Root mean square error of GPFU

G_i Radius criteria of the ith GPFU

H_i Hypersphere of the ith GPFU

$\mathbf{K}^i$ Shape matrix of the ith GPFU

M Number of output units or number of PFUs

N Dimension of input pattern or number of teaching patterns

P Performance index of GPFN

$\mathbf{R}'$ $(N \times (M+1)$ matrix defined by $[\mathbf{r}_1', \mathbf{r}_2', \cdots, \mathbf{r}_N']^t$

S_N N dimensional sample space

S_Y Subspace spanned by $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M$

$\mathbf{Y}$ $N \times M$ matrix defined by $[\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M]^t$

$\mathcal{D}(\mathbf{p}_i)$ Domain of $\mathbf{p}_i$

$\mathcal{L}(\mathbf{Y}^k)$ Linear mainfold formed by the linear combination of the selected M vectors, $\mathbf{y}_i^k$, $i = 1, \cdots, M$ of $\mathbf{Y}^k$

$\mathcal{R}(\mathbf{y}_i)$ Range of $\mathbf{y}_i$

α Decaying factor for $\partial E_p / \partial \mathbf{n}_j$

β Increment rate of ρ_i

ε_m Error margin

ρ_i Parameter saturation criteria for the jth output unit

λ_i The ith eigenvalue of cross-correlation matrix Σ

ψ Potential function

ψ_i The ith potential function

σ_0 Initial marginal standard deviation

σ_j the jth marginal standard deviation of the ith GPFU

ϕ Potential field

ϕ_i The ith potential field

ϕ_{ji} The jth actual output value for the pth teaching pattern

Λ_p Measuring quantity defined by $\min_i \lambda_i$

Σ Cross-correlation matrix defined by $1/N \sum_{i=1}^{N} \mathbf{r}_i' \mathbf{r}_i'^t$