Paper

Overlapped Multi-Neural-Network and Its Training Algorithm

Member	Jinglu HU	(Kyushu University)
Member	Kotaro HIRASAWA	(Kyushu University)
Non-member	Qingyu Xiong	(Kyushu University)

This paper presents an overlapped multi-neural-network (OMNN). An OMNN consists of two parts: main part and partitioning part. The main part, structurally, is the same as an ordinary feedforward neural network, but it is considered as one consisting of several subnets. All subnets have the same input-output units, but some different hidden units. The partitioning part divides input space into several parts, each of which is associated with one subnet. An improved random search algorithm called RasID is introduced to train the OMNN. Numerical simulations show that such an OMNN has superior performance in that it has better presentation ability than an ordinary neural network and better generalization ability than a non-overlapped multi-neural-network.

Keywords: Neural networks, multiple models, overlap, brain-like model, random search

1. Introduction

In an ordinary neural network, individual units do not have any special relations with the input patterns. However, according to recent knowledge of brain science, it is suggested that there exists function localization in a human brain, which means that specific neurons are activated corresponding to certain sorts of sensory information the brain receives [1]. Therefore, a brain-like neural network should have the capabilities of function localization as well as learning. Such a brain-like model may be more efficient because its individual units are mainly used to remember certain input patterns. To obtain such a brain-like model, the main problem is how to guide a training algorithm to realize the function localization. Learning Petri Network [2] is a brain-like model. However, it is rather difficult to train it, further study is needed.

In this paper, we consider a simple implementation of such brain-like model by using a so called overlapped multi-neural-network (OMNN). An OMNN consists of two parts: main part and partitioning part. The main part, structurally, is the same as an ordinary feedforward neural network; but it is considered as one consisting of a class of subnets; all the subnets have the same input-output units but some different hidden units. The partitioning part divides input space into several parts, each of which is associated with one subnet. Various clustering or classification methods and algorithm may be used to implement the partitioning part. As an example, we use a competitive network that has the same number of outputs as the number of parts of the input space divided. Each output of competitive network represents one input space part. For an input pattern, only one of competitive network outputs gives 1, and only nodes of the subnet associated with this output are fired, while all other nodes remain inactive. This realizes function localization of OMNN.

On the other hand, viewed from a viewpoint of multiple network, the main part of OMNN is a multiple network with overlapped units; when the number of overlapped units is zero, it becomes an ordinary multiple network, while it is an ordinary feedforward neural network when the number of overlapped units is equal to the total hidden units. In this sense, an OMNN can be seen as a learning network between an ordinary feedforward neural network and an ordinary multiple network. Figure 1 shows an image of such relationship. Moreover, it is well-known that multimodel approach is based on a divide-and-conquer strategy [3] [4]. The bias-variance trade-off is an important issue for the divide-and-conquer strategy. In general, dividing the training data set into subsets for process identification tends to increase the variance and decrease the bias. In multimodel approach, soft or fuzzy splits of data are often used to ease the bias-variance dilemma [5] [6]. In the OMNN, the overlap of units between subnets is expected to have the same functions.

Backpropagation (BP) is the most popular algorithm to train an ordinary neural network. However, it has been found that it is difficult to apply a BP to train efficiently our OMNN, especially for batch training, because in an OMNN different input patterns correspond to different subnets. To get around this difficulty, we introduce a random search algorithm called RasID. RasID is first presented in Refs. (7) (8), but in this paper we have made some modifications to the RasID so as to improve its efficiency.

This paper is organized as follows: Section 2 introduces the overlapped multi-neural-network; Section 3 describes the training algorithm; Section 4 carries out some numerical simulations; Section 5 gives some discussions on OMNN; Finally Section 6 presents conclusions.

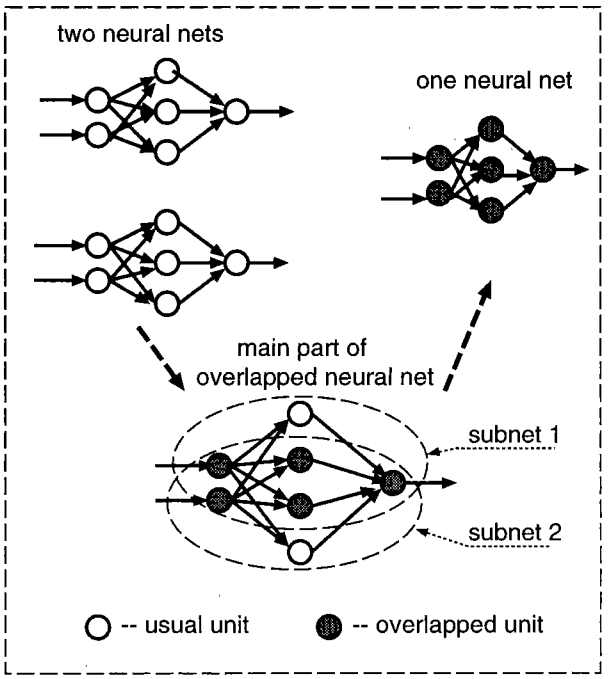


Fig. 1. Multiple network, overlapped-multi-network and ordinary feedforward network

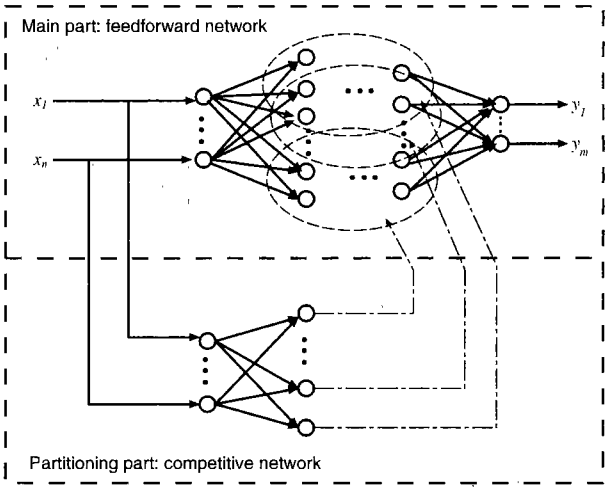


Fig. 2. An OMNN consisting of two parts: main part and partitioning part.

2. Overlapped Multi-Neural-Network

An OMNN has the capabilities of both learning and function localization. As shown in Fig.2, it consists of two parts: main part and partitioning part. The main part realizes the capability of learning and the partitioning part classifies the input space so as to realize the capability of function localization.

2.1 Partitioning Part The role of this part is partitioning of operating region. Let us consider problems such as system identification and pattern recognition. The operating region is defined as $\mathcal{Z}$. An operating point $\boldsymbol{z} \in \mathcal{Z}$ is a vector of variables. The operating region is partitioned into M operating regimes $\mathcal{Z}_i$ $(i = 1, ..., M)$ which is a subset of $\mathcal{Z}$, on the basis of certain prior knowledge. The input and output vectors of the model are called $\boldsymbol{x}$ and $\boldsymbol{y}$ and consist of n and m different variables respectively, where $\boldsymbol{x} = [x_1\, x_2 \ldots x_n]$ and $\boldsymbol{y} = [y_1\, y_2 \ldots y_m]$. Various clustering or classification methods and algorithms may be used for the partitioning, see e.g., (3) (9). But in our OMNN, we also use neural networks for this part. Candidates are the networks of competitive learning and self-organizing maps [10] [11]. In our simulations, we simply use competitive learning network.

As shown in the lower part of Fig.2, the competitive learning network has one layer of input neurons and one layer of output neurons. An input pattern $\boldsymbol{x}$ is a sample point in the n-dimensional real vector space. Binary-valued (1 or 0) *local representations* are used for the output nodes. That is, there are as many output neurons as the number of classes (M) and each output node represents a pattern category. The network serves the important role of selecting winner, via a competitive learning process, highlighting the "winner-take-all" schema. That is, the output unit receiving the largest input is assigned a value of 1, whereas all other units are suppressed to a 0 value.

2.2 Main Part The main part, structurally, is an ordinary feedforward multi-layer neural network, but it is considered to consist of M overlapped subnets. If we denote the set of input units by $\mathcal{I}$, the set of output units by $\mathcal{O}$ and the set of ith hidden layer units by $\mathcal{N}_i$ $(i = 1, 2, ...)$, then the jth subnet can be described by $\{\mathcal{I}, \mathcal{S}_{1j}, \mathcal{S}_{2j}, \ldots, \mathcal{O}\}$ where the set of ith hidden layer units, $\mathcal{S}_{ij}$, is a subset of $\mathcal{N}_i$. That is, $\mathcal{S}_{ij} \subset \mathcal{N}_i$ $(j = 1, 2, ..., M)$. These M subnets are associated with the M operating regimes. For the input and output vectors $\{\boldsymbol{x}, \boldsymbol{y}\}$ of operating regime $\mathcal{Z}_j$, only the units corresponding to the jth subnet of OMNN are active, while all other units are inactive and have zero output. The sets of hidden layer units of subnets, $\mathcal{S}_{ij}$, are determined based on the prior knowledge used in operating region partition such that all hidden layer units available in the sets $\mathcal{N}_i$ are used in subsets $\mathcal{S}_{ij}$.

There are several parameters to be determined: the number of subnets; the number of hidden units for each subnet; the number of hidden units that overlap or the number of total hidden units. The values of these parameters are certainly problem depended. However, how to determine them is still left as an open problem. Only some heuristic results are available, further research is needed.

(1) *The number of subnets.* It is equal to the number of parts of input space divided. In many cases, prior knowledge is available for determining the number of input space to be divided. According to our exprience on several classification and system identification problems when there is no prior knowledge available, dividing the input space into 4 to 6 parts gives better results.

(2) *The number of hidden units for each subnet.* This depends on the complexity of each part of input space. When no prior knowledge is available, the same number may be used for all subnets. It is found that when the number of hidden units for

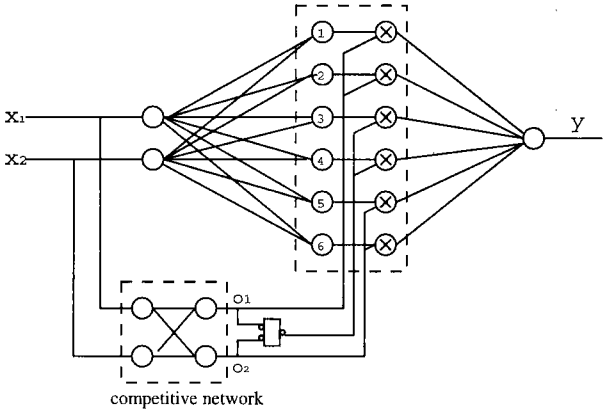


Fig. 3. An example of OMNN showing the relationship between partitioning part and main part.

each subnet is equal to $\frac{1}{3}$ to $\frac{2}{3}$ of the total number of hidden units, OMNN gives better results.

(3) *The number of hidden units that overlap.* This is rather difficult to determine. It seems that it is easier to determine the total number of hidden units first, then the number of hidden units that overlap by assigning the number of hidden units for each subnet based on the '$\frac{1}{3}$ to $\frac{2}{3}$' rule.

2.3 An Example Here we give a simple example to show the relation between the partitioning part and the main part in an OMNN. As shown in Fig.3, in the example, the main part has 2 input units, 6 hidden units and 1 output unit. It is divided into two subnets with the same input-output units; the first subnet contains the 1st to 4th hidden units, and the second subnet contains the 3rd to 6th hidden units. The partitioning part is a competitive network containing two output units. It divides the input space into two parts. The outputs of the competitive network control the firing of the hidden units of main part. When an input set from the 1st part of input space appears, the competitive network gives $O_1 = 1$ and $O_2 = 0$. This fires hidden units 1 to 4, while hidden units 5 and 6 contribute 0 to the output of OMNN. When an input set is from the 2nd part, then only hidden units 3 to 6 are fired, and hidden units 1 and 2 will kept inactive. From this example, it is clear that OMNN has not only learning capability, but also function localization capability.

2.4 OMNN Training OMNN training consists of two steps: the training of partitioning part and the training of main part. The former usually must be done before carrying out the latter training. In this paper, a competitive network is used as partitioning part. A competitive learning algorithm is therefore used for the training. In our simulations, we simply use the competitive learning algorithm provided by Matlab Neural network toolbox [12]. We here only discuss training of the main part.

The main part, structurally, is a feedforward neural network. Similar to ordinary neural network training, the training of the main part of OMNN is formulated as a nonlinear optimization defined by

$$\Theta = \arg\min_{\Theta}\{E\}, \qquad \Theta \in \mathcal{W} \cdots\cdots\cdots\cdots (1)$$

where E is the criterion function, Θ is the parameter vector and $\mathcal{W}$ denotes a compact region of parameter vector space. Let $\hat{\boldsymbol{y}}$ is the OMNN output corresponding to the input vector $\boldsymbol{x}$. Then the criterion function E is defined by

$$E = \sum_{i\in\mathcal{D}} (||\boldsymbol{y}(i) - \hat{\boldsymbol{y}}(i)||) \cdots\cdots\cdots\cdots\cdots\cdots (2)$$

where $\mathcal{D}$ is the set of training data.

In neural network literature, back propagation (BP) algorithm is the most popular one for network training. However, such a popular BP algorithm is not easy to be applied to solve (1) efficiently, especially for batch training. The reason is that the input vectors from different operating regimes correspond to different subnets of OMNN, hence different gradient computations. To get around this difficulty, we shall develop a modified random search algorithm for OMNN training.

3. Training Algorithm

3.1 Basic Structure of Algorithm The random search algorithm is modified from an ordinary random search method [13][14] and was first presented in Refs (7) (8). But some improvements have been made to increase its efficiency.

Let $\Theta(k) = [\lambda_1(k), ..., \lambda_l(k), ...]^T$ be the parameter vector $\Theta \in \mathcal{W}$ given the value of the k-th search, and $\Delta\Theta(k)$ be the random vector $\Delta\Theta(k) = [\Delta\lambda_1(k), ..., \Delta\lambda_l(k), ...]^T$ generated based on a probability density functions (PDFs) after the k-th search. Then the random search algorithm can be described as follows.

Step 1: Choose an initial value $\Theta(0) \in \mathcal{W}$, calculate $E(\Theta(0))$ and set $k = 0$;

Step 2: Generate a random search vector $\Delta\Theta(k)$. If $\Theta(k) + \Delta\Theta(k) \notin \mathcal{W}$, then let $\Theta(k+1) = \Theta(k)$ and go to **Step 3**, else

- calculate $E(\Theta(k) + \Delta\Theta(k))$. If $E(\Theta(k) + \Delta\Theta(k)) < E(\Theta(k))$, the current search is said to be success and then set $y^{(k)} = 1$ and $\Theta(k+1) = \Theta(k) + \Delta\Theta(k)$, else
- calculate $E(\Theta(k) - \Delta\Theta(k))$. If $E(\Theta(k) - \Delta\Theta(k)) < E(\Theta(k))$, the current search is said to be success too and then set $y^{(k)} = 1$ and $\Theta(k+1) = \Theta(k) - \Delta\Theta(k)$, otherwise
- the search is said to be failure and then set $y^{(k)} = 0$ and

$$\Theta(k+1) = \begin{cases} \Theta(k) & \text{If } k_{er}^{+} > k_{er} \\ & \text{and } k_{er}^{-} > k_{er} \\ \Theta(k) + \Delta\Theta(k) & \text{If } k_{er}^{+} < k_{er}^{-} \\ \Theta(k) - \Delta\Theta(k) & \text{If } k_{er}^{+} \geq k_{er}^{-} \end{cases}$$

where $k_{er} \geq 1$ is the maximum error ratio, k_{er}^{+} and k_{er}^{-} are defined by

$$k_{er}^{+} = \frac{E(\Theta(k) + \Delta\Theta(k))}{E(\Theta(k))},$$

$$k_{er}^{-} = \frac{E(\Theta(k) - \Delta\Theta(k))}{E(\Theta(k))}.$$

Step 3: Stop if pre-specified conditions are met, else set $k = k + 1$ and go to **Step 2**.

3.2 Strategy for Generating $\Delta\lambda_l$ In a conventional random search algorithm, $\Delta\lambda_l$ is usually generated by using a Gaussian probability density function (PDF). It is reported that the search efficiency is improved by tuning the average and dispersion of Gaussian PDF [13][14]. However, it is found that the improvement is very limited since the range of tuning the average and dispersion of Gaussian PDF can not be very large [8][15].

To solve this problem, in the modified random search algorithm we introduce a new sophisticated PDF that is more tunable, and then develop a scheme tuning the PDF so as to improve searching efficiency of the algorithm by controlling the searching range and directions using local information.

3.2.1 New Sophisticated PDF The new PDF is defined by

$$f(\Delta\lambda_l) = \begin{cases} (1-q_l)\beta_l e^{\beta_l \Delta\lambda_l} & \text{If } \Delta\lambda_l \le 0 \\ q_l \beta_l e^{-\beta_l \Delta\lambda_l} & \text{If } \Delta\lambda_l > 0 \end{cases} \quad \cdots(3)$$

where $q_l \in [0,1]$ and β_l are two adjustable parameters. It follows that a random search variable $\Delta\lambda_l$ is generated by

$$\Delta\lambda_l = \begin{cases} \frac{1}{\beta_l} \ln\left(\frac{z_l}{1-q_l}\right) & \text{If } 0 < z_l \le 1-q_l \\ -\frac{1}{\beta_l} \ln\left(\frac{1-z_l}{q_l}\right) & \text{If } 1-q_l < z_l < 1.0 \end{cases} \quad \cdots\cdots(4)$$

where z_l's are random values uniformly distributed between 0 and 1.

As shown in Fig.4, the parameter β_l can be used to control the local search range (the variance of search variable Δ_l). The larger the β_l is, the smaller the local search range will be. On the other hand, the parameter q_l can be used to control the search probability in positive or negative direction. The larger the q_l is, the higher the search probability in positive direction is. Typically, when $q_l = 0.5$ there is the same search probability in positive direction and in negative direction. In this way, we are able to realize an efficient searching in network training by determining the parameters β_l and q_l based on the past success-failure information.

3.2.2 Adaptively Determining β_l β_l controls the search range; the smaller β_l is, the higher the possibility to generate a large random search variable. In the algorithm β_l is determined by the following heuristic rule

$$\beta_l = \underline{\beta} + \left(\bar{\beta}_l - \underline{\beta}\right) e^{-\phi I_{sf}} \quad \cdots\cdots(5)$$

where $\underline{\beta}$ is a constant, and $\bar{\beta}_l$, ϕ and I_{sf} are three adjustable indexes.

- Generally speaking, parameter λ_l ($l = 1, 2, ...$) have different scales and hence need different tuning sensitivities. We use $\bar{\beta}_l$ to determine the tuning sensitivity of individual search variables. The larger the β_l is, the smaller the tuning sensitivity will be. It is clear that when a search fails in both positive and negative searching directions, the tuning sensitivity in these directions should be reduced, otherwise, it may be increased. Let us tune $\bar{\beta}_l$ in the following way

$$\bar{\beta}_l = \begin{cases} \bar{\beta}_l + s_i|\Delta\lambda_l| & \text{if } y^{(k)} = 0 \\ \bar{\beta}_l - s_d|\Delta\lambda_l| & \text{if } y^{(k)} = 1 \\ \bar{\beta}_{l0} & \text{if k=pre-specified integers} \end{cases}$$

where s_i, s_d are two coefficients assigned appropriate positive numbers with $s_i \ge s_d$, and $\bar{\beta}_{l0}$ is initial value of $\bar{\beta}_l$. When no useful prior knowledge is available, $\bar{\beta}_{l0} = \bar{\beta}_0$ is usually used, where $\bar{\beta}_0$ is an appropriate value.

- In the area where it is possible to find a good solution locally, an intensified search is performed. Inspired by the scheme used in the evolution strategy [16], the intensified search is performed with keeping the success-failure ratio P_{sf}, calculated in a moving window, close to a pre-assigned value P_{sf0}†. This is realized by fixing I_{sf} and adapting ϕ in the following way

$$\phi = \begin{cases} c_i\phi & \text{If } P_{sf} > P_{sf0} \text{ and } \phi > \phi_{\min} \\ \phi & \text{If } P_{sf} = P_{sf0} \text{ or } \phi \le \phi_{\min} \\ c_d\phi & \text{If } P_{sf} < P_{sf0} \text{ and } \phi > \phi_{\min} \\ \phi_0 & \text{If } k = \text{pre-specified integers} \end{cases}$$

where $c_i \ge 1.0$, $0 < c_d \le 1.0$ are two coefficients assigned appropriate values, ϕ_0 and $\phi_{\min}$ are the initial value and the minimum value of ϕ, respectively.

- When the intensified search finds the best solution locally, the success-failure ratio P_{sf} becomes small. Consequently ϕ is reduced to $\phi \le \phi_{\min}$. Then a diversified search is carried out in order to escape from the local minimum. This is realized by adjusting I_{sf} heuristically in the following way

$$I_{sf} = \begin{cases} I_{sf0} & \text{If } (\phi > \phi_{\min}) \text{ or } (y^{(k)} = 1 \text{ and } I_{sf} > I_{sf\max}) \text{ or } k = \text{pre-specified integers} \\ I_{sf} - \triangle I_{sf1} & \text{If } y^{(k)} = 1 \text{ and } \phi \le \phi_{\min} \\ I_{sf} + \triangle I_{sf2} & \text{If } y^{(k)} = 0 \text{ and } \phi \le \phi_{\min} \end{cases}$$

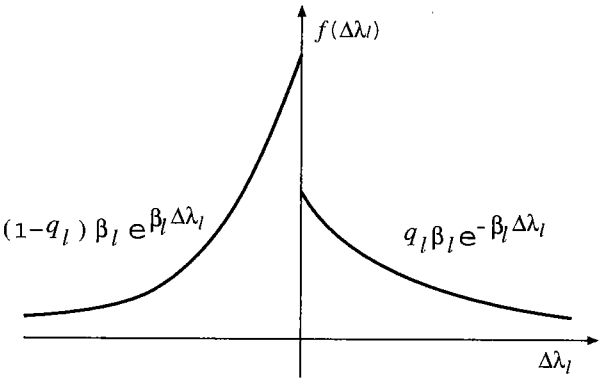


Fig. 4. New sophisticated probability density function that is more tunable.

†The success-failure ration, P_{sf}, is defined as $P_{sf} = \frac{n_s}{n_f}$ where n_s is the number of success in the past L searches, and n_f the number of failure.

where I_{sf0} is the initial value of I_{sf}, $\triangle I_{sf1}$, $\triangle I_{sf2}$ are two appropriate positive values with $\triangle I_{sf1} \leq \triangle I_{sf2}$. We can see from the above rule that I_{sf} is fixed on the initial value I_{sf0} during an intensified searching period, while increases gradually in the diversified search. When $I_{sf} > I_{sf\max}$ and $y^{(k)} = 1$ becomes true, it is supposed that a prospective area is found, then I_{sf} is set to I_{sf0} and another intensified search will be performed. In the above rules, $\bar{\beta}_l$, ϕ and I_{sf} are reset to their initial values periodically, which is introduced to increase the robustness of algorithm. The period can be set to a rather large integer, for instance, 50,000.

3.2.3 Adaptively Determining q_l The algorithm can be greatly speeded up if the probability of searching for prospective areas increases, especially in the case where the searching space $\mathcal{W}$ is large. In Ref. (14), this is realized to some extent by adapting the mean of random search vector $\boldsymbol{x}$ based on the past success-failure information. In our scheme, the same function will be realized more efficiently by adapting the parameter q_l. Let us first give q_l an initial value of 0.5 and adjust it only when the past search is success ($y^{(k)} = 1$) in the following way

$$q_l = \begin{cases} \alpha q_l & \text{If } \Delta\lambda_l(k) < 0 \\ q_l & \text{If } \Delta\lambda_l(k) = 0 \\ \alpha q_l + (1-\alpha) & \text{If } \Delta\lambda_l(k) > 0 \end{cases}$$

where $\alpha \in (0, 1]$ is an appropriate value.

The rule shows that q_l is adjusted such that the searching probability on the direction of the past successful search is increased. Statistically, the success-failure of the past search describes the trend information in the surface of criterion function to some extent. It therefore can be considered as useful information for improving the searching performance.

3.2.4 Typical Values of Parameters The algorithm discussed in this section contains several parameters which should be determined *a priori*. Fortunately, most of these parameters are not very sensitive to problems. Table 1 shows a set of typical values of these parameters, see Ref. (8) for a discussion of how to determine those parameters.

4. Simulation Studies

4.1 Simulations on RasID Since the popular BP algorithm can not be applied efficiently to the OMNN training, we instead developed a random search algorithm called RasID. In Refs. (8) (15), RasID is compared with the conventional random search that uses Gaussian PDF [14]. It is found that RasID has better searching efficiency. The main reason is that RasID uses a more tunable PDF instead of Gaussian PDF.

We here use a popular object function called Goldstein and Price function [17] to illustrate the intensified search and the diversified search in RasID. The function is a well-known benchmark problem, for which a gradient-based algorithm, e.g. BP, usually be stuck at a local minimum. We will show that RasID can escape from local minima and find the global minimum. The

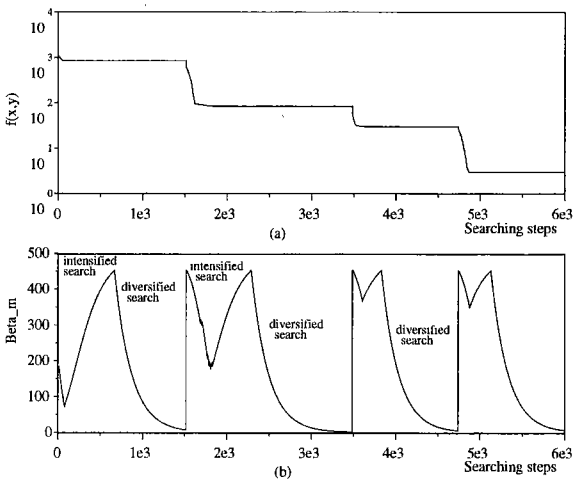


Fig. 5. Simulation results of the Goldstein and Price Function: (a) value of function $f(x, y)$, (b) value of parameter β_l

Goldstein and Price function is described by

$$f(x,y) = [1 + (x+y+1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + (2x-3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)] \cdots\cdots (6)$$

The parameters of RasID used are shown in Table 1 except that $\beta_1 = 100$, $\triangle I_{sf2} = 0.5$, $s_i = 0$ and $s_d = 0$. We modified the parameters because this is a rather simple problem. We have run the algorithm many times and show the worst cases, in the sense that more local minima reached. The results are shown in Fig.5.

From Fig.5, we can see that the searching begins with an intensified search. At the beginning of an intensified search, the success-failure ratio is large, thus β_l is adjusted to be smaller so that the local search range becomes larger in order to speed up the search. When the algorithm comes near a local minimum, the success-failure ratio becomes small, then β_l is adjusted to be larger so that the local search range is smaller in order to have an intensive search. When better solutions can not be obtained, it is considered that the algorithm reaches a local minimum. The algorithm then begins a diversified search so as to escape from the local minimum. After the escaping succeeds, another intensified search begins. In this way, the algorithm not only can find global minimum definitely, provided sufficient searching steps, but also has satisfied convergence properties.

4.2 Simulations on OMNN Let us consider a benchmarch problem: Two Nested Spirals. The task is to use OMNNs to separate two nested spirals.

The training sets consist of 152 associations formed by assigning the 76 points belonging to each of the nested spirals to two classes. This is a nontrivial classification task, which has been extensively used as a benchmark for evaluation of neural network training [18]. We use the example to discuss generalization ability and representation ability of OMNN.

The OMNN used in the simulations is denoted by $N_{n-r-m} \times M/n_T$ where N_{n-r-m} is a subnet with n

Table 1. Typical values of parameters

$\beta 0$	β_1	α	ϕ_0	$\phi_{\min}$	I_{sf0}	P_{sf0}	k_{er}
0.1	2000	0.995	0.1	0.001	10	0.3	1.0001
$\triangle I_{sf1}$	$\triangle I_{sf2}$	$I_{sf\max}$	c_i	c_d	s_i	s_d	
0.02	0.1	100	1.01	0.995	5	10	

input units, r hidden units and m output units, M is the number of parts of input space divided, and n_T is the total number of hidden layer units. Since all subnets have the same number of hidden units, obviously when $n_T = r * M$, that is, there is no overlapped units, the OMNN becomes a multi-neural-network, and when $n_T = r$ the OMNN is equivalent to an ordinary feedforward neural network.

4.2.1 Generalization Ability In multi-model approach, soft or fuzzy splits of data are often used to ease the bias-variance dilemma [5][6]. In this simulation, we will show that overlapping hidden units has the same role so that it improves generalization ability of multi-neural-network.

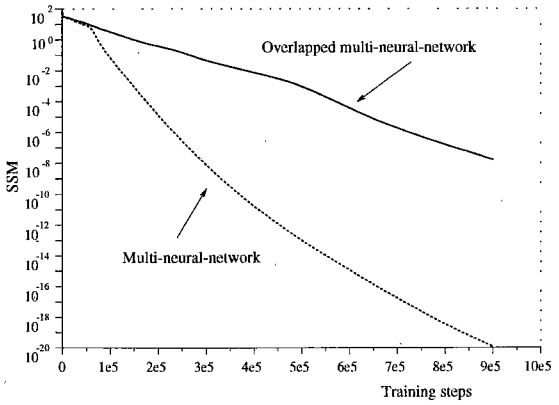
Let $M = 3$, $n_T = 30$, $r = 10$, $n = 2$, and $m = 1$. Then we have a multi-neural-network with 3 subnets. Using a competitive learning algorithm, the competitive network is trained so that the input space is divided into 3 parts. The symbols $\bigoplus$ in the Fig.6(b) and (c) show the feature vectors of three parts. Then we use the algorithm described in Section 3 to train the OMNN. The dashed line on Fig.6(a) shows the sum squares error for training data. We can see that the training error is very small, but the performance of classification is not satisfied because some areas are misclassified, see Fig.6(b).

Next, we reduce $n_T = 30$ to $n_T = 22$, but keep $r = 10$. That is, the three subnets overlap. Subnet 1 overlap with Subnet 2, Subnet 2 overlaps with Subnet 3, each of which has 4 overlapped units. The OMNN is then trained with the same algorithm. The solid line in Fig.6(a) shows the training curve and Fig.6(c) show the classification results. We can see that in the network with overlapping, although the training error becomes larger, the performance of classification is improved. This means that the overlap in an OMNN improves the generalization ability. The results seem easy to understand because the latter OMNN is more compact with fewer total number of hidden units than the former OMNN. The former one has 30 hidden units, while the latter has only 22.

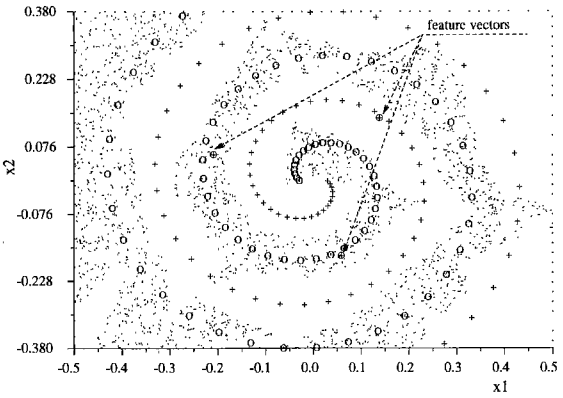
4.2.2 Representation Ability In this simulation, we will show that the OMNN has better representation ability than an ordinary feedforward neural network. We will reduce the number of total hidden units n_T to a value such that an ordinary neural network with this number of hidden layer units is not able to solve the problem. In this way, we are able to illustrate the role of function localization in an OMNN.

It is found from simulation studies that a neural network with one hidden layer of less than 16 units failed to solve the above two-nested-spiral problem. We therefore let $n_T = 15$, $M = 6$, $n = 2$, $m = 1$, and r varying from 5 to 15. Obviously, when $r = 15 = n_T$ the OMNN reduces to an ordinary neural network that will be too small to solve the two-nested-spiral problem.

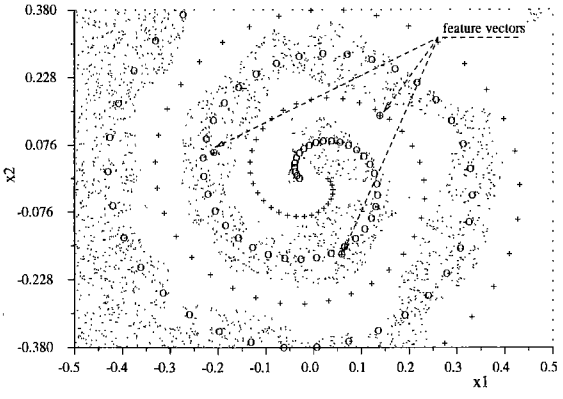
Similar to the former simulation, the competitive net-



(a) Sum squares error (SSE) for training data



(b) Result using non-overlapped multi-neural-network



(c) Result using overlapped multi-neural-network

Fig. 6. Comparisons of the OMNN to non-overlapped multiple net

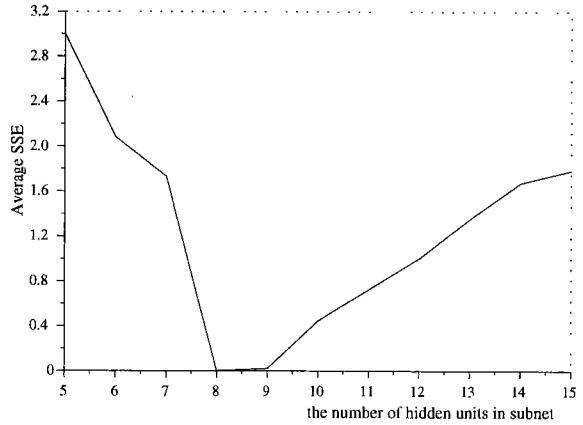
Table 2. Sum squares error (SSE) of OMNNs consisting of different subnets and different overlapped units

$N_{2-r-1} \times 6/15$	$r=5$	$r=6$	$r=7$	$r=8$	$r=9$	$r=10$	$r=12$	$r=13$	$r=14$	$r=15$
Average SSE	3.005	2.083	1.729	0.003	0.021	0.448	1.003	1.351	1.670	1.783

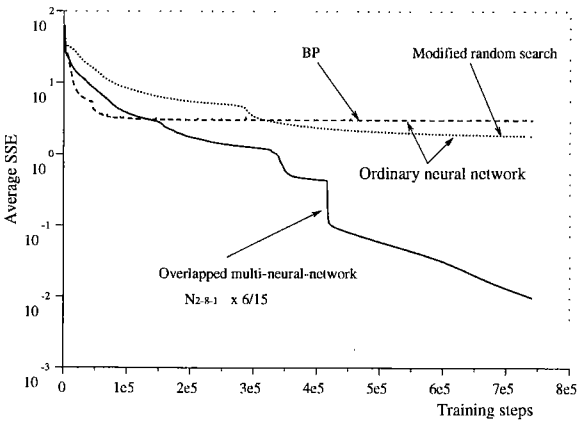
work is first trained by using a competitive learning algorithm. The competitive network has 6 output units, which divides the input space into 6 parts. Then for each given $r = 5$ to 15, we train the main part of OMNN three times with random initial values for 750,000 steps each time, and then average the results. Table 2 and Fig.7(a) shows the sum square errors (SSE) of various cases. Since the number of total hidden units and the number of of the parts of input space divided are fixed, the larger the number of subnet hidden units, the more the overlapped units. From Tab.2 and Fig.7(a), we can see that the OMNNs whose subnets have 8 or 9 hidden units have the best representation ability. When the number of subnet hidden units is 15, the same as the number of total hidden units, the OMNN becomes an ordinary neural network. An optimal OMNN has obviously better representation ability than an ordinary neural network with the same size.

Figure 7(b) compares the results of the ordinary neural network ($r = 15$, dotted line and dashed line) to the OMNN whose subnets have 8 hidden units ($r = 8$, solid line). It shows that the optimal OMNN has smaller training error than the ordinary neural network. On the other hand, since for an ordinary neural network, it is easy to apply backpropagation (BP) algorithm, we also show the result of a fast BP with momentum implemented by using Matlab neural network toolbox (trainbpx.m) [12]. Comparing the results of BP and the modified random search algorithm, RasID, shows that RasID used in this paper is efficient and comparable to an ordinary BP.

We have applied the OMNN to several other problems of system identification and classification. We found that when the input space is divided into 4 to 6 parts, and the subnet hidden layers have $\frac{1}{2}$ to $\frac{2}{3}$ of the total units of the corresponding hidden layers, the OMNNs have better representation ability. This result is interesting because it means that function localization in a neural network may improve the efficiency of individual neurons, hence improve the performance of neural network.



(a) SSE for different OMNN



(b) SSE for OMNN and ordinary neural network

Fig. 7. Results of OMNNs consisting of different subnets and different overlapped units

5. Discussions

One of distinctive features of a brain-like model to an ordinary neural network is that it has function localization capability. Simulation results show that function localization may improve the efficiency of individual neurons, hence the efficiency of the whole network.

5.1 Key Problems It is clear that a key problem in a brain-like model training is how to guide the training algorithm to realize function localization. In this paper, we introduced a competitive network for the purpose and manually realized the function localization. However, how to automatically realize the function localization is still left as an open problem.

5.2 Possible Solutions In order to realize the function localization, two steps are needed to consider. One is to extract the features of the problem from the available input-output sets. The second is to localize these features in the brain-like network. Self-organizing map (SOM) networks seem to be suitable for this purpose. This will be investigated in our future research.

6. Conclusions

Neural network is a simplified brain model which can learn any nonlinear mappings. However, human brain has the capability of function localization as well as the capability of learning. An ordinary neural network does not have the function localization ability. Inspired by recent knowledge of brain science, we try to develop a brain-like model that has both function localization capability and learning capability. This paper discusses a

simple implementation of such brain-like model: overlapped multi-neural-network. The structure and the learning algorithm have been discussed. The results of simulation studies show that function localization in a neural network improves the efficiency of individual neurons. Our future study will be directed toward to developing more sophisticated function localization scheme that may extract and then incorporate prior knowledge from input-output patterns.

(Manuscript received April 12, 2000, revised September 19, 2001)

References

(1) 澤口 俊之, 知性の脳構造と進化, 海鳴社, 1989.

(2) K. Hirasawa, M. Ohbayashi, S. Sakai, and J. Hu, "Learning Petri network and its applications to non-linear system control", *IEEE Trans on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 28, no. 6, pp. 781–789, 1998.

(3) Tor Arne Johansen, *Operation Regime Based Process Modeling and Identification*, PhD thesis, University of Trondheim, Norway, 1994.

(4) A.S. Weigend, M. Mangeas, and A.N. Srivastava, "Nonlinear gated experts for time series: Discovering regimes and avoiding overfitting", *International Journal of Neural Systems*, vol. 6, pp. 373–399, 1995.

(5) C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford, Uk: Oxford University Press, 1995.

(6) B. Eikens and M.N. Karim, "Process identification with multiple neural network models", *Int. Journal of Control*, vol. 72, no. 7/8, pp. 576–590, 1999.

(7) 平澤 宏太郎·東郷 和幸·胡 敬炉·大林 正直·邵 寧· 村田 純 一, "ニューラルネットワークの適応的ランダム探索最適化手法ー RasID ー", 計測自動制御学会論文集, vol. 34, no. 8, pp. 1088–1096, 1998.

(8) J. Hu, K. Hirasawa, and J. Murata, "RasID– Random search for neural networks training", *Journal of Advanced Computational Intelligence*, vol. 2, no. 4, pp. 134–141, 1998.

(9) T.A. Johansen and B.A. Foss, "ORBIT - operating regime based modeling and identification toolkit", *Control Engineering Practice*, vol. 6, pp. 1277–1286, 1998.

(10) T. Kohonen, *Self-Organization and Associative Memory, 2nd Edition*, Springer-Verlag, 1987.

(11) T. Kohonen, *Self-Organizing Maps*, Heidelberg:Springer, 1995.

(12) H. Demuth and M. Beale, *Neural Network Toolbox: for use with MATLAB*, The MATH WORKS Inc., 2000.

(13) J. Matyas, "Random optimization", *Automation and Remote Control*, vol. 28, pp. 244–251, 1965.

(14) F.J. Solis and J.B. Wets, "Minimization by random search techniques", *Mathematics of Operations Research*, vol. 6, pp. 19–30, 1981.

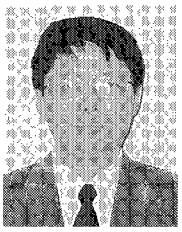
(15) J. Hu, K. Hirasawa, J. Murata, M. Ohbayashi, and Y. Eki, "A new random search method for neural network learning -RasID-", in *Proc. of IEEE International Joint Conference on Neural Networks (Alaska)*, 5 1998, pp. 2346–2351.

(16) I. Rechenberg, *Evolutions Strategie: Optimierung Technisher Sysytem Nach Prinzipieren der Biologischen Evolution*, Stuttgart: Frommann-Holzbooz, 1973.

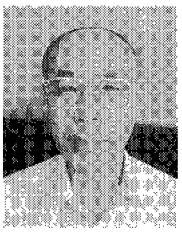
(17) L.C.W. Dixon and G.P. Szegö, Eds., *Towards Global Optimization 2*, North Holland, 1978.

(18) S.A. Solla and M. Fleisher, "Generalization in feedforward neural networks", in *Proc. of the IEEE International Joint Conference on Neural Networks (Seattle)*, 1991, pp. 77–82.

Jinglu HU (Member) He received the M.Sci. degree in 1986 from Zhongshan University, China and the Ph.D degree in 1997 from Kyushu Institute of Technology. From 1986 to 1993, he was a Research Associate and then a Lecturer in Zhongshan University. Since 1997, he has been Research Associate at Kyushu University. His current research interests are learning network theory, system identification and their applications. Dr. Hu is a member of the Society of Instrument and Control Engineers.



Kotaro HIRASAWA (Member) He received the M.S. degree in Electrical Engineering from Kyushu University in 1966. From April 1966, he served in Hitachi Lab. of Hitachi Ltd., and in 1989 he was a vice president of Hitachi Lab.. From August 1991 to November 1992, he served in Omika Factory of Hitachi Ltd.. Since December 1992, he has been a professor in the faculty of Engineering, Kyushu University. Now he belongs to the Graduate School of Information Science and Electrical Engineering, Kyushu University. Dr. Hirasawa is a member of the Society of Instrument and Control Engineers, a member of IEEE.



Qingyu Xiong (Non-member) He received the Master degree in 1991 from Chongqing University, China. Since April, 1999, he has been a doctor course student in Kyushu University.