A sum-of-product neural network (SOPNN)∗

Chun-Shin Lin*, Chien-Kuo Li

Department of Electrical Engineering, University of Missouri-Columbia, Columbia, MO 65211, USA

Received 21 September 1998; accepted 22 March 1999

Abstract

This paper presents a sum-of-product neural network (SOPNN) structure. The SOPNN can learn to implement static mapping that multilayer neural networks and radial basis function networks normally perform. The output of the neural network has the sum-of-product form

\[ \sum_{i=1}^{N_p} \prod_{j=1}^{N_v} f_{ij}(x_j), \]

where \( x_j \)'s are inputs, \( N_v \) is the number of inputs, \( f_{ij}() \) is a function generated through network training, and \( N_p \) is the number of product terms. The function \( f_{ij}(x_j) \) can be expressed as \( \sum_{k} w_{ijk} B_{jk}(x_j) \), where \( B_{jk}(\cdot) \) is a single-variable basis function and \( W_{ijk} \)'s are weight values. Linear memory arrays can be used to store the weights. If \( B_{jk}(\cdot) \) is a Gaussian function, the new neural network degenerates to a Gaussian function network. This paper focuses on the use of overlapped rectangular pulses as the basis functions. With such basis functions, \( W_{ijk} B_{jk}(x_j) \) will equal either zero or \( W_{ijk} \), and the computation of \( f_{ij}(x_j) \) becomes a simple addition of some retrieved \( W_{ijk} \)'s. The structure can be viewed as a basis function network with a flexible form for the basis functions. Learning can start with a small set of submodules and have new submodules added when it becomes necessary. The new neural network structure demonstrates excellent learning convergence characteristics and requires small memory space. It has merits over multilayer neural networks, radial basis function networks and CMAC in function approximation and mapping in high-dimensional input space. The technique has been tested for function approximation, prediction of a time series, learning control, and classification. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Neural networks; Function approximation; Sigma–Pi network; Sum-of-product neural network; Memory-based neural network

∗This research was partially supported by the National Science Foundation under the Grant ECS-9408857 and by the Research Board of the University of Missouri.

*Corresponding author. Tel.: +1-573-882-3520; fax: +1-573-882-0397.

E-mail address: linc@missouri.edu (C.S. Lin)
1. Introduction

This paper presents a sum-of-product neural network (SOPNN) that can learn to implement static mapping in a similar manner to that of multilayer neural networks and the radial basis function networks. The output of the neural network has the sum-of-product form \( \sum_{i=1}^{N_v} \prod_{j=1}^{N_v} f_{ij}(x_j) \), where \( x_j \)'s are inputs, \( N_v \) is the number of inputs, \( f_{ij}(\cdot) \) is a function generated through network training, and \( N_p \) is the number of product terms. The SOPNN has some resemblance to the Sigma–Pi network [8,15,5,6]. Both use product and summation units. However, the SOPNN uses flexible-form self-generated functions instead of polynomials.

The new structure overcomes difficulties in function approximation and mapping in high-dimensional input space, encountered in multilayer neural networks (MNNs) [15,7,9,18] and radial basis function networks (RBFNs) [11,19,3,16,12,4,17,2,20,1]. It is well known that, when the input dimension is high and the desired mapping is complicated, it is hard to predict how long the learning process of an MNN will take and whether the learning will converge to an acceptable result. Another type of neural network, RBFN, often uses Gaussian function as the basis function. Since a Gaussian function provides function mapping to a local area, the learning convergence is quick and hence the RBFN has less problems when compared to that of an MNN. In addition, learning is likely to only alter local information and thus, will be less likely to destroy previously learned information. However, the number of basis functions may become enormous for problems with a large number of input variables. To increase the fitting power of each basis function in order to reduce the number of basis functions, some researchers suggested making the Gaussian function scaleable in each dimension and rotatable in the input space [19,3]. The trade-off is the increased learning difficulty.

The SOPNN has \( f_{ij}(x_j) \) calculated in a form as \( \sum_{k=1}^{l+n-1} w_{ijk} B_{jk}(x_j) \), where \( B_{jk}(\cdot) \) is a single-variable basis function, \( W_{ijk} \)'s are weight values stored in memory, \( l \) is the quantized element number for \( x_j \), and \( n \) is the number of basis functions in the neighborhood used for storing information for \( x_j \). If all \( B_{jk}(\cdot) \)'s are Gaussian functions, the new neural network degenerates to a Gaussian function network. Although \( B_{jk}(\cdot) \) could be any adequate basis function, in this paper, we will focus on the use of overlapped rectangular pulses. With such basis functions, \( W_{ijk}B_{jk}(x_j) \) will equal either zero or \( W_{ijk} \), and the computation of \( f_{ij}(x_j) \) becomes a simple addition of retrieved \( W_{ijk} \)'s. In this structure, \( \prod_{j=1}^{N_v} f_{ij}(x_j) \) can be viewed as a self-generated basis function. However, it is not necessary to be in any specific form and this makes the “basis function” very flexible. The new neural network structure will solve the extensive memory requirement problem as well as the learning difficulty existent in currently available types of neural networks.

In Section 2, the new structure is presented. Section 3 gives the learning rules and procedure. Section 4 examines the capability of the new structure in function approximation, prediction, classification and control applications. Section 5 gives conclusions.
2. The new neural network structure

Fig. 1 shows the SOPNN structure. The network output \( SOPNN() \) is the sum of the outputs from a set of submodules. As introduced in Section 1, the output of the neural network has the sum-of-product form \( \sum_{i=1}^{N} \prod_{j=1}^{N} f_{ij}(x_j) \) and the function \( f_{ij}(x_j) \) is expressed in a form as \( \sum_{k=1}^{n} w_{ijk} B_{jk}(x_j) \) where \( B_{jk}(\cdot) \) is a single-variable basis function. In this study, the focus is on a memory-based structure that uses overlapped rectangular pulses as basis functions. With such basis functions, only \( W_{ijk} \)'s need be stored and linear memory arrays are ideal for this purpose.

Fig. 2 illustrates the arrangement of the overlapped rectangular pulse functions, and the computation of \( f_{ij} \) given an input \( x_j \). Each input variable \( x_j \) is divided into \( N_e \) elements and the \( n_e \) neighboring elements are grouped into a block. Each block is assigned a basis function, which can be a bell-shape function, a cubic spline function, a triangular or a rectangular pulse function. Using the rectangular pulse function, each element will be covered by \( n_e \) blocks. The computation of \( f_{ij}(x_j) \) is the addition of weights associated to the \( n_e \) blocks covering the specific \( x_j \). The arrangement makes one element learn and it also alters the weight values of its neighborhood. This gives the SOPNN its generalization capability. As shown in Fig. 2, there are \( N_e + n_e - 1 \) blocks. Thus the memory size for each variable in each submodule equals

![Fig. 1. The SOPNN neural network structure.](image-url)
Fig. 2. Blocks for overlapped rectangular pulses and the computation of $f_{ij}$ ($n_e = 4$ in this illustration).

\[ f_{ij}(x_i \text{ in element } 4) = \text{sum of weights for blocks 4, 5, 6 and 7} \]

$N_e + n_e - 1$, which is usually small (typically 20 to 200). For $N_p$ submodules and $N_v$ variables, the total memory size required would be $N_p N_v (N_e + n_e - 1)$. A memory size of 20 k, which is very small, is enough for a structure with 20 submodules, 10 input variables and 100 blocks (i.e., $N_e + n_e - 1 = 100$). Utilizing a small memory makes the scheme easy to implement and very attractive.

$N_e$ determines the resolution of the quantized input space. A structure with a larger $N_e$ can provide a more accurate representation, but requires a large memory (the size is $N_p \times N_v \times (N_e + n_e - 1)$). Typically, a value between fifty and a couple hundred will be adequate. The number of blocks, $N_e + n_e - 1$, affects the generalization capability in learning. Using larger blocks (fewer blocks) improves the learning speed and generalization, but reduces the approximation accuracy. The number of available training samples could be a consideration factor in selection of the block size. The number of blocks should not be greater than the number of training samples in order to guarantee that over-fitting will not occur. One reasonable suggestion is to have it less than one-tenth of the number of training samples. How many submodules are needed is problem-dependent; a more complicated mapping requires more submodules. Fortunately, the algorithm to be introduced in the next section can add submodules during the learning. No pre-determination of the number is necessary.

3. Neural network learning

3.1. Learning rules

Contents of the memory arrays are adapted during the learning phase. The gradient decent learning rule can be derived and used for learning. For a given sample, the cost
function to be minimized can be the squared error

\[ E = \frac{1}{2} \varepsilon^2 = \frac{1}{2} (y_t - SOPNN)^2 = \frac{1}{2} \left( y_t - \sum_i P_i \right)^2, \] (1)

where \( \varepsilon \) is the network output error, \( y_t \) is the target output value for this training sample, and \( P_i \) indicates the output of submodule \( i \).

One can adjust \( W_{ijk} \)'s to reduce the cost function. The learning rule based on gradient descent should be

\[ \Delta W_{ijk} = -\alpha \frac{\partial E}{\partial W_{ijk}} = -\alpha \frac{\partial E}{\partial P_i} \frac{\partial P_i}{\partial f_{ij}} \frac{\partial f_{ij}}{\partial W_{ijk}}. \] (2)

Since the output of submodule \( i \) is

\[ P_i = f_{i1} f_{i2} \ldots f_{iN_v}, \] (3)

the partial derivative of \( P_i \) with respect to \( f_{ij} \) is

\[ \frac{\partial P_i}{\partial f_{ij}} = \prod_{k \neq j} f_{ik}. \] (4)

Thus

\[ \frac{\partial P_i}{\partial f_{ij}} \frac{\partial f_{ij}}{\partial W_{ijk}} = \left\{ \prod_{p \neq j} f_{ip} \right\} B_{jk}(x_s^{(p)}), \] (5)

where \( x_s^{(p)} \) denotes the \( j \)th element of a given input vector \( s \). The incremental weight update, \( \Delta W_{ijk} \) (in the \( j \)th memory array of submodule \( i \)) in Eq. (2) becomes

\[ \Delta W_{ijk} = \alpha \left\{ \prod_{p \neq j} f_{ip} \right\} B_{jk}(x_s^{(p)}) \varepsilon, \] (6)

where \( \alpha \) is a learning rate. With rectangular pulse basis, only \( n_x B_{jk}(x_s) \)'s have nonzero values. Thus updating will occur only for those \( n_x \) corresponding memory elements (i.e., weights).

3.2. Learning procedure

The neural network size may be predetermined or determined during the learning phase. The two different arrangements are referred to as the fixed structure and the flexible structure. The following summarizes a learning procedure in which the neural network structure can grow automatically to an adequate size. However, one can start
from an adequate initial size network and not allow the network to grow. Then the procedure is the same as a fixed structure.

1. Initialize the neural network with one submodule (one may start with more than one submodule). Select all learning parameters.
2. Initialize all memory arrays of the new submodule with random memory contents between $-\delta$ and $\delta$.
3. Obtain a training sample.
4. Compute the overall output for this sample and calculate the error.
5. Use $\Omega\%$ of the error to update the new submodule and $\left(100 - \Omega\right) / K\%$ of the error for each of the old submodules, where $K$ is the current number of old submodules.
6. Update all memory arrays using Eq. (6).
7. If the error for the past $N_1$ samples is under a specified level, then stop.
8. If the improvement in the last $N_2$ samples is insignificant (for instance, the reduction of error is less than 3%), then add one more submodule to the neural network and go to step 2. Otherwise, go to step 3.

$N_1$ in step 7 and $N_2$ in step 8 will be numbers selected by the user. The value $\Omega$ in step 5 determines the weight used when updating the memory contents in the new and old submodules. When $\Omega$ equals to 100%, the learning in all old submodules is disabled. By using unequal backpropagated error terms one utilizes new submodules more fully and this speeds up the learning process. One reasonable suggestion is to have $\Omega$ equal to 50%. This makes a large update in the weight in the new submodule during the training. Note that at the very beginning without any old submodule, the error should be the same for all initial submodules.

4. Evaluation of the new structure for different applications

In this section, the technique is evaluated for different applications including function approximation, prediction, learning control, and classification. In the prediction example, the neural network size is automatically determined during the learning process.

4.1. Function approximation

In this subsection, results for approximating the following six-variable nonlinear function using the new SOPNN technique are presented:

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = \sin(\pi x_1) \sin(\pi x_2) \exp(-x_1^2 - x_2^2)$$
$$+ \sin(\pi x_3) \sin(\pi x_4) \exp(-x_3^2 - x_4^2)$$
$$+ \sin(\pi x_5) \sin(\pi x_6) \exp(-x_5^2 - x_6^2)$$
$$- 1 \leq x_1, x_2, x_3, x_4, x_5, x_6 \leq 1.$$  (7)
A fixed structure with three submodules was used in this experiment. Each input variable was quantized into 122 elements with 12 elements forming a block. There were six variables and each submodule was composed of six memory arrays. This network required 2592 memory locations. For training, 10,000 training patterns were randomly generated in each epoch. The value of each variable was restricted to be in the range \([-1, 1]\). The learning rate \(\alpha\) was set to 0.1. After 10 epochs, the sum of squared errors (SSE) had dropped to 3.59 for the last 10,000 training patterns. Fig. 3 shows the SSE curve. The error dropped quickly to a low level after just one epoch.

Multilayer neural networks (MNNs) have also been tested for a comparison with the new SOPNN structure. Three-layer feedforward neural networks with 40, 50 and 60 hidden neurons were trained using the error backpropagation procedure. With different initial weights, the training either took a very long time or failed. The best result was from the structure with 60 hidden neurons. As in the previous illustration for the SOPNN training, 10,000 training patterns were also used in each epoch. This MNN was trained for 9000 epochs until the SSE dropped to 89.78. Fig. 4 shows the

![Fig. 3. The SSE curve for the SOPNN training.](image1)

![Fig. 4. The SSE curve for the MNN training.](image2)
SSE curve. For smaller MNNs with 10 hidden units, the SSE remains at a value above 500 and cannot be reduced to an acceptable level.

To compare the approximation results, we plot the target function generated from Eq. (7), the output from the SOPNN, and the output from the MNN in Fig. 5. While there are six inputs, four of them are fixed to obtain these plots; $x_2$, $x_4$, $x_5$ and $x_6$ are set to 0.5. The SOPNN demonstrates an excellent approximation.

The outputs of the three submodules are shown in Fig. 6. Note that the output of the SOPNN in Fig. 5(b) is the sum of these three submodule outputs.

Fig. 5. Plots for (a) the function $f$ in Eq. (7), (b) the result of the trained SOPNN and (c) the result of the MNN, with $x_2$, $x_4$, $x_5$ and $x_6$ set to 0.5.
Another function with higher nonlinearity has been suggested for testing by one reviewer. The function is
\[ f(x_1, x_2, x_3, x_4, x_5, x_6) = \sin(\pi x_1 x_2) \exp(-x_1^2 - x_2^2) + \sin(\pi x_3 x_4) \exp(-x_3^2 - x_4^2) + \sin(\pi x_5 x_6) \exp(-x_5^2 - x_6^2). \]
With six submodules, the SSE can be reduced from 2301.87 in the beginning to 10.40 and 1.025 at the 207th and 220th epochs, respectively. The results indicate that more submodules and a longer training time are needed due to the higher nonlinearity of the function. This example has been also used to show performance using different \( \Omega \) values, including 50, 75, and 100%, during the learning. In our simulation, we started with four submodules. The SSE dropped from the initial value of 2266.56 to 31.49 in 34 epochs. The fifth submodule was then added. Fig. 7(a) shows the subsequent learning curves for different \( \Omega \) values. The sixth submodule was added later at epoch 55. Fig. 7(b) shows...
Fig. 7. Learning curves for different values of $\Omega$: (a) Learning curves upon adding the fifth submodule at epoch 35; (b) Learning curves upon adding the sixth submodule at epoch 55.

The rest of the learning curves. Note that by having $\Omega$ equal 100%, the old submodules will not be further adjusted. Only the new submodule will learn and this reduces the learning capability of the SOPNN. Fig. 7 shows that the final errors for the case with $\Omega$ equal to 100% are larger. In this example, 75% for the value of $\Omega$ seems to give the best performance. The error converges faster. In this case, four or five submodules share 25% error in learning. That means the updating in the new submodule was about 12 to 15 times heavier than that in the old submodules. This enables major changes in the new submodule and in the meantime allows minor adjustments in old submodules.
4.2. Prediction

The Mackey–Glass (MG) time series [13,10] was used to evaluate the capability of the new structure for prediction. The Mackey–Glass equation represents a model for white blood cell production in leukemia patients. It mimics nonlinear oscillation in physiological processes. The Mackey–Glass delay-difference equation is shown below:

\[ y(k + 1) = (1 - b)y(k) + a \frac{y(k - \tau)}{1 + y^{10}(k - \tau)}, \]  

where \( a = 0.2, b = 0.1, \) and \( \tau = 17. \)

This model is complicated due to the addition of a time delay \( \tau \) in the nonlinear equation. The state space diagram in Fig. 8 shows the quasiperiodic nature of the time series. The objective here is to model the time series and to predict the value of the time series at some future time, based on four previous values.

In the experiment, four measurements \( y(k), y(k - 6), y(k - 12) \) and \( y(k - 18) \) were used to predict \( y(k + 1) \). A thousand data points were generated for usage – the first 500 for training and the subsequent 500 for testing. The SOPNN structure was allowed to grow during the training. Each submodule in the neural network was composed of four single-input memory arrays. Each input variable was quantized into 122 elements with 12 elements forming a block. We started with two submodules. The network was trained until the improvement on SSE in two consecutive epochs became insignificant. A new submodule was then added. The learning rate \( \alpha \) was set to 0.2 and \( \Omega \) was set to 50%. The procedure continued until the SSE for 500 training data dropped under 0.018 after 81 epochs. The structure grew to a size of four submodules. Fig. 9 shows the SSE curve for the entire learning procedure.

As stated in the previous paragraph, 500 training data and 500 testing data had been generated for experimental usage. Fig. 10(a) shows these 1000 data (the solid line)

---

Fig. 8. The state space diagram showing the quasiperiodic nature of the MG time series.
as well as the prediction result from a well-trained SOPNN (the dash-line). While the prediction is very accurate, two lines are very close and the dashed line is hardly observable. In the experiment, inputs to the SOPNN for making prediction were generated from the Mackey–Glass difference equation. This emulated the use of measured data. One may call this one-step prediction; all information before and at the time instant \( k \) is available for predicting the output value at time instant \( k + 1 \). The prediction for the 500 testing data (data points 501 to 1000) is very accurate. Fig. 10(b) shows the error.

The ability of long-term prediction has also been examined. For long-term prediction, the output of the SOPNN is fed back to the network inputs for calculating future values. Fig. 11 shows the result for the long-term prediction error. Note that although the prediction curve has a very similar shape as the target curve, some small delay causes the observable errors in the plot.

The multilayer neural network has also been tested for a comparison. A three-layer feedforward neural network with 35 hidden nodes was used in the experiment. The learning rate was set to 0.1. The network was trained using the error backpropagation (BP) algorithm for 29,000 epochs until the SSE dropped to 0.05. Fig. 12 shows the prediction error using “measured” data. However, Fig. 13 shows that the multilayer neural network is unable to make long-term prediction using the feedback data from the neural network model itself. This can be explained using Fig. 8. That figure shows the region where training patterns locate. For long-term prediction, the output of the SOPNN at time \( k \) is feedback to the input for computing the output for the next time instant \( k + 1 \). It seems that the error is accumulated, and the data point quickly moves outside of the nominal region and gets stuck at some place; this induces large errors. This is a phenomenon somehow similar to the behaviour of an unstable system.
4.3. Control

In this subsection, the SOPNN is applied to the identification and control of a nonlinear dynamic system. Fig. 14 shows the model reference adaptive control (MRAC) structure used in this part of study.

In this MRAC technique, a reference model is selected and the controller should adapt to make the plant output the same as the output of the reference model. In the structure in Fig. 13, SOPNN$_1$ and SOPNN$_2$ will learn to model the plant and implement the controller, respectively.
The selected plant is a single-input and single-output system with unknown dynamics described by the following nonlinear difference equation:

\[ y(k + 1) = g[y(k), y(k - 1), y(k - 2), u(k), u(k - 1)] \tag{9} \]

where \( y(k) \) is the current output and \( u(k) \) is the current control input. The unknown function \( g \) has the form

\[
g(x_1, x_2, x_3, x_4, x_5) = \frac{x_1x_2x_3x_5(x_3 - 1) + x_4}{1 + x_2^2 + x_3^2}. \tag{10} \]
Fig. 13. The error of long-term prediction using the MNN. The well-trained MNN is unable to perform long-term prediction. (prediction starts at \( k = 501 \)).

Fig. 14. Model reference adaptive control with SOPNNs. This is one example used by Narendra and Parthasarathy [14]. The neural network structure SOPNN, for identifying the plant has six submodules with each submodule composed of five memory arrays. Each input variable is quantized into 82 elements with 10 elements forming a block. The SOPNN was trained for 240,000 time steps with the learning rate \( \alpha \) set equal to 0.06 and a random input signal uniformly distributed in the range \([-1, 1]\).

On-line learning for developing the controller starts after a good plant model is generated. The controller SOPNN has \( y(k), y(k - 1), u(k - 1) \), and \( r(k) \) as inputs for generating the control input \( u(k) \). One may express the function as

\[
u(k) = SOPNN_2[y(k), y(k - 1), u(k - 1), r(k)]\]
The SOPNN$_2$ has six submodules of which each consists of four memory arrays. Each variable is quantized into 82 elements with 10 elements forming a block. The input $r(k)$ is selected to be

$$r(k) = 0.5\sin(2\pi k/250) + 0.1\sin(2\pi k/25).$$

Since $r(k)$ varies slowly with time, the reference model has been simply selected to be a single-period delay, i.e., $y^*(k + 1) = r(k)$. The value $y^*(k + 1)$ is the desired output in the MRAC technique. The controller was trained on-line with the learning rate $\alpha$ set equal to 0.01. During the training, the weights of the neural controller SOPNN$_2$ were adjusted to reduce the control error $y^* - y_p$. The error was backpropagated through the plant model (SOPNN$_1$) into the controller (SOPNN$_2$) for weight adjustment. After 300,000 time steps on-line training, the performance of the controller was tested. Fig. 15 shows the error in control tracking.

4.4. Classification

The SOPNN has also been tested for classification, which the multilayer neural network is most suitable for. For testing, four classes were created using two 5-variable functions. These two functions are

\[ g_1(x_1, x_2, x_3, x_4, x_5) = x_2x_3 \exp[(x_3x_4 - x_5)^2] \]
\[ - 2(x_1x_4 - 1)^2x_5 + x_3 - x_4 - 0.4, \]  \hspace{1cm} (11)

\[ g_2(x_1, x_2, x_3, x_4, x_5) = x_2x_3(2x_4x_5 - 1) - \sin(1.5\pi x_1) - 1.0, \]  \hspace{1cm} (12)

where $0 \leq x_1, x_2, x_3, x_4, x_5 \leq 2.0$.

Fig. 15. Output error with SOPNN$_2$ as the controller. (Control output range is in $[-0.6, 0.6]$.)
Fig. 16. Distribution of four classes with \( x_3 = x_4 = 1.0 \).

\[ g_1(x_1, x_2, x_3, x_4, x_5) = 0 \] and \( g_2(x_1, x_2, x_3, x_4, x_5) = 0 \) are used as boundaries for four classes:

- class I: \( g_1 \geq 0 \) and \( g_2 \geq 0 \),
- class II: \( g_1 \geq 0 \) and \( g_2 < 0 \),
- class III: \( g_1 < 0 \) and \( g_2 \geq 0 \),
- class IV: \( g_1 < 0 \) and \( g_2 < 0 \).

Fig. 16 shows the distribution of four classes on the \( x_1 - x_2 \) plane with the other three variables set to 1.0.

The SOPNN with six submodules was used for learning the classification. Each variable was quantized into 82 elements with each block composed of 10 elements. Thirty thousand training samples were generated evenly within the input domain. The learning rate \( \alpha \) was selected to be 0.01. After 30 epochs of training, 3000 patterns were tested. The SOPNN correctly classified 93.7% of test patterns. With the patterns randomly generated in the input domain, there will be always some falling very close to the boundary and being misclassified. The performance will be much better if the classes are well separated.

The multilayer neural network has also been tested for a comparison. A three-layer feedforward neural network with 20 hidden nodes was used in the experiment. The MNN was trained using the error backpropagation (BP) algorithm for 5000 epochs with the learning rate set to 0.1. The MNN correctly classified 93.1% of test patterns. The accuracy is about the same as that of the SOPNN but the learning time is much longer.

5. Conclusion

A sum-of-product form has been developed and evaluated for various applications. The novel SOPNN structure is a memory-based neural network that can self-generate
the necessary basis functions. Since the memory cost has been reduced and the memory technology has improved in the past decade, practical implementation of the proposed structure is inexpensive. It is possible to increase the neural network size by adding new submodules during the learning process. This makes the guess of the number of submodules unnecessary. The SOPNN can be viewed as a hybrid structure that combines the table lookup technique (such as CMAC) and the computation-based technique (such as MNN and RBFN). A table lookup technique heavily relies on data memorization and requires very little computation. A hybrid structure such as the SOPNN overcomes the problem of requiring large memory, which exists in table lookup structures such as CMAC (especially for high-dimensional modelling). Our experiments show that learning converges easily; this is an important merit of the SOPNN compared to multilayer neural networks. The major computation required in SOPNN includes multiplication and addition. More expensive computation of exponential function value, which is typically required in MNN and radial basis function nets, is not involved in SOPNN.

Acknowledgements

Suggestions from reviewers have helped in improving the quality of the paper. Their efforts are greatly appreciated.

References


Chun-Shin Lin received his PhD from the School of Electrical Engineering at Purdue University in 1980. He worked as a research assistant and later a research associate for the Lab of Applications for Remote Sensing at Purdue University from 1979–1981. He was an associate research fellow for the Institute of Information Science at Academia Sinica from 1981–1984, and an associate Professor at the San Diego State University from 1984–1987. He is currently an associate professor with the Department of Electrical Engineering at the University of Missouri-Columbia. His past research areas included robotics, neural networks, image processing, evolutionary programming, missile trajectory, sensor validation, and adaptive and learning systems. He has published more than 70 technical articles in referred journals and conference proceedings. He served as a technical editor for IEEE Transactions on Robotics and Automation in 1991–1993, as committee members for many conferences in robotics and neural network areas. He was once an adjunct associate professor with the National Taiwan University, and had been a visiting professor for the Institute of Industrial Technology in two summers. He worked with NASA Lewis Research Center, Air Force Research Labs, and NASA Johnson Space Center on research during the Summers in 1992–1998.

Chien-Kuo Li received the B.S. degree in control engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1989, and the M.S. and Ph.D. degrees in electrical engineering from the University of Missouri, Columbia, MO, in 1994 and 1997, respectively. Currently, he is an assistant professor in the Department of Information Management at Shih Chien University, Taipei, Taiwan. His research interests are in the areas of neural networks, evolutionary programming, signal processing, and intelligent control.