Neural Processing Letters **17:** 1–19, 2003.

A Sigma-Pi-Sigma Neural Network (SPSNN)

CHIEN-KUO LI
Department of Information Management, Shih Chien University, Taipei, Taiwan, ROC
e-mail: ckli@mail.usc.edu.tw

Abstract. This letter presents a sigma-pi-sigma neural network (SPSNN) structure. The SPSNN can learn to implement static mapping that multilayer neural networks and radial basis function networks usually do. The output of the SPSNN has the sum of product-of-sum form $\sum_{n=1}^{K}\prod_{i=1}^{n}\sum_{j=1}^{N_v} f_{nij(x_j)}$, where x_j's are inputs, N_v is the number of inputs, $f_{nij}()$ is a function to be generated through the network training, and K is the number of pi-sigma network (PSN) which is the basic building block for SPSNN. A linear memory array can be used to implement $f_{nij}()$. The function $f_{nij}(x_j)$ can be expressed as $\sum_{k=1}^{N_q+N_e-1} w_{nijk}B_{ijk}(x_j)$, where $B_{ijk}()$ is a single-variable basis function, w_{nijk}'s are weight values stored in memory, N_q is the quantized element number for x_j, and N_e is the number of basis functions in the neighborhood used for storing information for x_j. If all $B_{ijk}()$'s are Gaussian functions, 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, $w_{nijk}B_{ijk}(x_j)$ will equal either zero or w_{nijk}, and the computation of $f_{nij}(x_j)$ becomes a simple addition of retrieved w_{nijk}'s. 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.

Key words. function approximation, memory-based neural network, ridge polynomial network, self-generated basis function, sigma-pi-sigma neural network

1. Introduction

This letter presents a sigma-pi-sigma neural network (SPSNN) 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 SPSNN has the sum of product-of-sum form $\sum_{n=1}^{K}\prod_{i=1}^{n}\sum_{j=1}^{N_v} f_{nij}(x_j)$, where x_j's are inputs, N_v is the number of inputs, $f_{nij}()$ is a function to be generated through the network training, and K is the number of pi-sigma network (PSN) which is the basic building block for SPSNN. The SPSNN has some resemblance to the ridge polynomial network (RPN) [25], which uses a special form of ridge polynomials. While the RPN uses polynomial terms, the SPSNN uses linear memory arrays that self-generate suitable basis function terms. Due to the flexibility, it is expected that the memory-based sigma-pi-sigma neural network should have more powerful modeling capability.

The new structure overcomes difficulties in function approximation and mapping in high-dimensional input space, encountered in multilayer neural networks (MNNs)

[12, 13, 20, 23, 26] and radial basis function networks (RBFNs) [3–6, 9, 11, 15, 16, 24, 27, 28]. 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 the MNNs will take and whether the learning will converge to an acceptable result. Another type of neural networks, RBFN, often uses the 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 the 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 [13, 23]. The trade-off is the increased learning difficulty.

The SPSNN has $f_{\mathrm{nij}}(x_{\mathrm{j}})$ calculated in a form as $\sum_{k=1}^{N_q+N_e-1} w_{nijk} B_{ijk}(x_j)$, where $B_{\mathrm{ijk}}()$is a single-variable basis function, w_{nijk}'s are weight values stored in memory, N_{q} is the quantized element number for x_{j}, and N_{e} is the number of basis functions in the neighborhood used for storing information for x_{j}. If all $B_{\mathrm{ijk}}()$'s are Gaussian functions, the new neural network degenerates to a Gaussian function network. Although $B_{\mathrm{ijk}}()$ could be any adequate basis function, in this paper, we will focus on the use of overlapped rectangular pulses. With such basis functions, $w_{nijk} B_{ijk}(x_j)$ will equal either zero or w_{nijk}, and the computation of $f_{\mathrm{nij}}(x_{\mathrm{j}})$ becomes a simple addition of retrieved w_{nijk}'s. In SPSNN, $\prod_{i=1}^{n} \sum_{j=1}^{N_v} f_{nij}(x_j)$ can be viewed as a *self-generated basis function*. However, it does not have 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, control and classification applications. Section 5 gives conclusions.

2. The New Neural Network Structure

The structure of a SPSNN is composed of different orders of pi-sigma networks (PSNs). The PSN is the basic building block for the SPSNN. Figure 1 shows a Kth order memory-based PSN. The network output $P_K(+)$ has the 'product-of-sum' form $\prod_{i=1}^{\mathrm{K}} \sum_{j=1}^{N_v} f_{\mathrm{K}ij}(x_j)$. Figure 2 shows a SPSNN whose output is the sum of outputs from K different orders of PSNs. The output SPSNN(·) equals $\sum_{n=1}^{\mathrm{K}} \prod_{i=1}^{n} \sum_{j=1}^{N_v} f_{nij}(x_j)$. As introduced in Section 1, the output of the neural network has the sum of product-of-sum form $\sum_{n=1}^{\mathrm{K}} \prod_{i=1}^{n} \sum_{j=1}^{N_v} f_{nij}(x_j)$ and the function $f_{\mathrm{nij}}(x_{\mathrm{j}})$ is expressed in a form as $\sum_{k=1}^{N_q+N_e-1} w_{nijk} B_{ijk}(x_j)$ where $B_{\mathrm{ijk}}()$ is a single-variable basis function. In this study, the focus is on a memory-based structure that uses overlapped

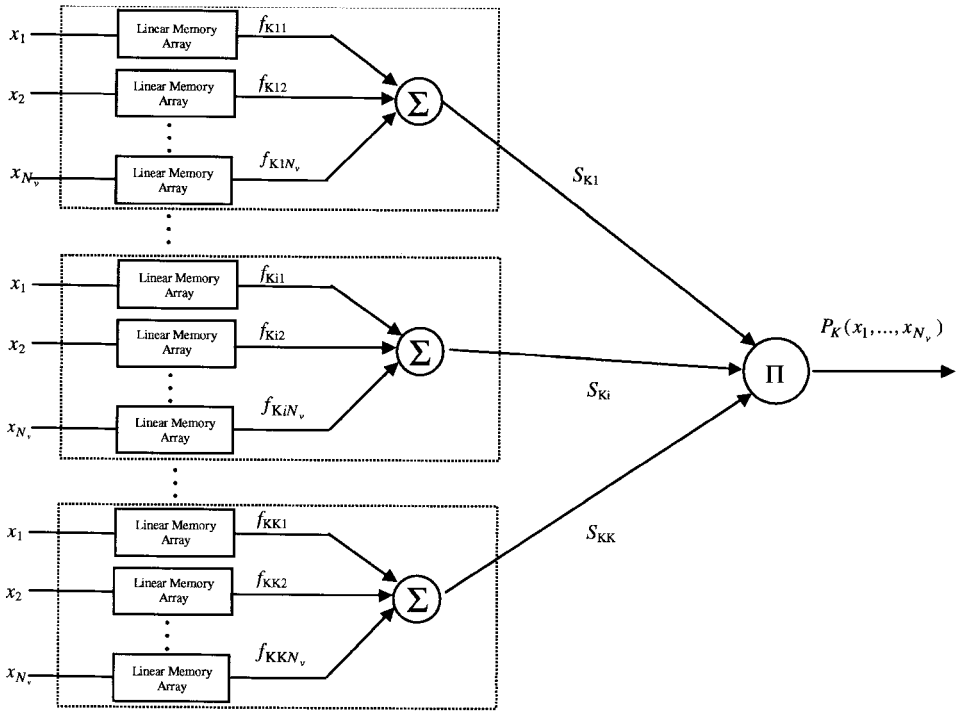


Figure 1. A basic building block (*K*th order PSN) for the SPSNN.

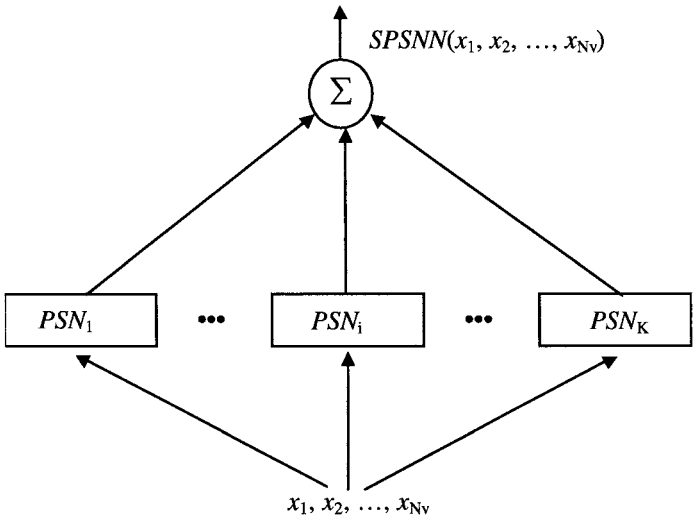


Figure 2. A *K*th order SPSNN is composed of K different orders of PSNs; PSN_i represents an *i*th order PSN as shown in Figure 1.

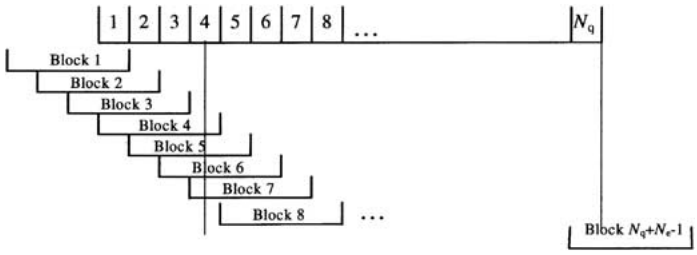


Figure 3. Blocks for overlapped rectangular pulses and the computation of f_{nij} ($N_e = 4$ in this illustration).

rectangular pulses as basis functions. With such basis functions, only w_{nijk}'s need to be stored and linear memory arrays are ideal for this purpose.

Figure 3 illustrates the arrangement of the overlapped rectangular pulse functions, and the computation of f_{nij} given an input x_{j}. Each input variable x_{j} is divided into N_q 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 is covered by N_e blocks. The computation of $f_{\text{nij}}(x_{\text{sj}})$ is the addition of weights associated to the N_e blocks covering the specific x_{sj}. The arrangement makes one element learn and it also alters the weight values of its neighborhood. This gives the SPSNN the generalization capability. As shown in Figure 3, there are $N_q + N_e - 1$ blocks. Thus the memory size for each variable equals $N_q + N_e - 1$, which is usually small (typically 20 to 200). For a K-th order SPSNN with N_v variables, the total memory size required will be $\frac{1}{2} \times K \times (K+1) \times N_v \times (N_q + N_e - 1)$. A memory size of 15 k, which is considered small, is enough for a 5th order SPSNN structure with 10 input variables and 100 blocks (i.e., $N_q + N_e - 1 = 100$). The requirement of a small memory makes this scheme easy to implement and very attractive.

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

3. Neural Network Learning

3.1. LEARNING RULES

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

$$E = \frac{1}{2}\varepsilon^2 = \frac{1}{2}(y_t - \text{SPSNN})^2 \tag{1}$$

where ε is the network output error, y_t is the target output value for the training sample. The following equations can be used to derive the learning rule for the Kth order SPSNN:

$$\text{SPSNN}(\cdot) = PSN_1 + PSN_2 + \cdots + PSN_{\text{n}} + \cdots + PSN_{\text{K}} \tag{2}$$

$$PSN_{\text{n}} = \prod_{i=1}^{n} S_{ni} \tag{3}$$

$$S_{ni} = \sum_{j=1}^{N_v} f_{nij}(x_j) \tag{4}$$

$$f_{nij}(x_j) = \sum_{k=1}^{N_q+N_e-1} w_{nijk} B_{ijk}(x_j) \tag{5}$$

The values of w_{nijk} can be adjusted to reduce the cost function. The learning rule based on gradient descent should be

$$\begin{aligned} \Delta w_{nijk} &= -\alpha \frac{\partial E}{\partial w_{nijk}} \\ &= -\alpha\varepsilon \frac{\partial \varepsilon}{\partial w_{nijk}} \\ &= \alpha\varepsilon \frac{\partial P_n}{\partial S_{ni}} \frac{\partial S_{ni}}{\partial f_{nij}} \frac{\partial f_{nij}}{\partial w_{nijk}} \end{aligned} \tag{6}$$

where α is a learning rate. The amount to be updated for w_{nijk} is

$$\Delta w_{nijk} = \alpha(y_t - \text{SPSNN}) \left\{ \prod_{p \neq i} S_{np} \right\} B_{ijk}(x_j^{(s)}) \tag{7}$$

where $x_j^{(s)}$ denotes the jth element of a given input vector $\mathbf{s}$. With the use of rectangular pulse basis function, only $N_e \text{B}_{ijk}(x_j^{(s)})$'s have nonzero values. Thus updating will occur only for those N_e corresponding memory elements (i.e., weights).

3.2. LEARNING PROCEDURE

The order of the SPSNN structure may be predetermined or determined during the learning. These 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 will grow to an adequate order. However, by predetermining the order of the SPSNN and not allowing size growing, the procedure is still applicable.

1. Initialize the neural network with the Kth-order SPSNN. Select all learning parameters.
2. Initialize all memory arrays of the newly added PSN (with an order $> K$) with random memory contents between $-\delta$ and δ.
3. Obtain a training sample.
4. Compute the overall output for this sample and calculate the error.
5. Use Ω % of the error to update the new PSN and (100-Ω)/N% of the error for each of the old PSN, where N is the current order of the old network.
6. Update all memory arrays using Equation (7).
7. If the error for the past N_1 samples is acceptable, 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 PSN (with an order higher than the current one by one) 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 numers selected by the users. The value Ω in step 5 determines the weight used in updating the memory contents in the new and old PSNs. When Ω equals 100, the learning in all old PSNs is disabled. By using unequal backpropagated error terms, we can fully utilize the newly added PSN and this speeds up the learning. One reasonable suggestion is to have Ω equal 50. This makes a large update in the weight in the new PSN during the training. Note that at the very beginning without any old PSNs, the error should be evenly distributed to all initial PSNs.

4. Evaluation of the New Structure for Different Applications

In this section, the new structure is evaluated for different applications including function approximation, prediction, learning control, and classification. All input variables in each of the application are divided into the same number of blocks. However, this is not always the case. For problems, some variables may have large domain, while some others may have smaller domain, the number of blocks should not be the same in all variables. In order to show the performance, the normalized mean squared error ($NMSE$) is measured. It is defined as

$$NMSE = \frac{\sqrt{MSE}}{\sqrt{\frac{\sum_{p=1}^{n}(y_{tp}-\bar{y})^2}{n}}} \tag{8}$$

where MSE is the mean squared error, y_{tp} is the target output value for sample input p, and $\bar{y}$ is the mean value of the target outputs. The dB value for $NMSE$ is defined as $20\log(NMSE)$.

4.1. FUNCTION APPROXIMATION

In this subsection, results for approximating a 2-D Gabor function [25] are provided for an illustration of the new SPSNN technique.

The convolution version of complex 2-D Gabor functions has the following form

$$g(x_1, x_2) = \frac{1}{2\pi\lambda\sigma^2} e^{-\{[(x_1/\lambda)^2 + x_2^2]/2\sigma^2\}} e^{2\pi i(u_0 x_1 + v_0 x_2)} \tag{9}$$

where λ is an aspect ratio, σ is a scale factor, and u_0 and v_0 are modulation parameters. In this simulation, the following Gabor function was used.

$$g(x_1, x_2) = \frac{1}{2\pi(0.5)^2} e^{-[(x_1^2 + x_2^2)/2(0.5)^2]} \cos(2\pi(x_1 + x_2)) \tag{10}$$

The training started with a second-order SPSNN. A third-order PSN was added after 75 epochs. The SPSNN shown in Figure 4 was used to approximate Equation (10).

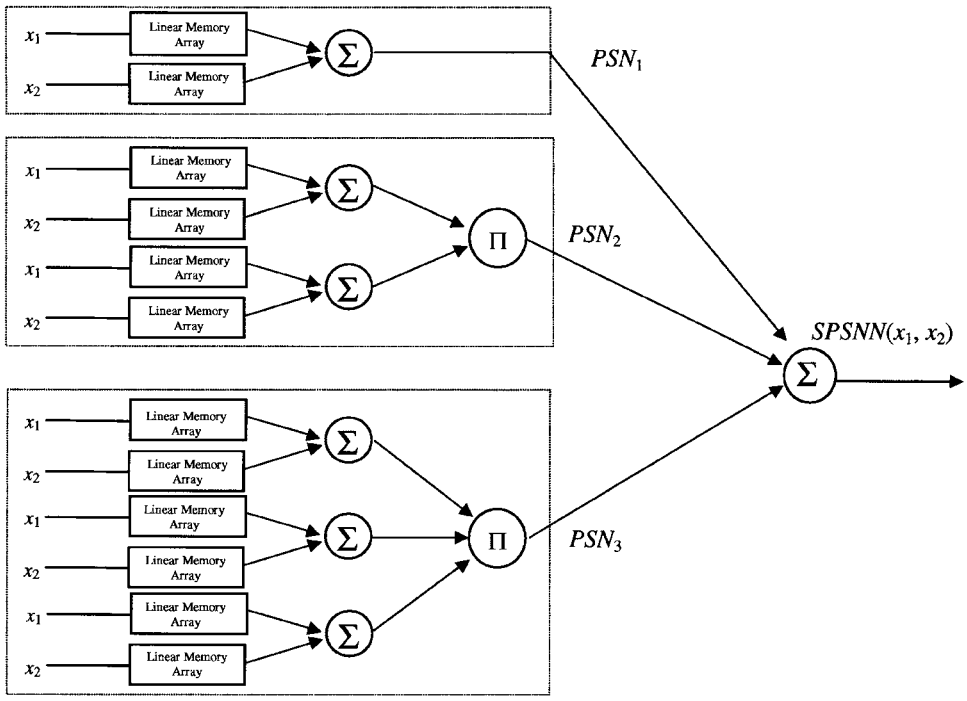


Figure 4. A third-order SPSNN used for approximating a Gabor function (10), where PSN_n is an nth order PSN.

Each input variable was quantized into 133 elements with 12 elements forming a block. The third-order SPSNN has 12 linear memory arrays and requires a memory size of 1728. The learning rate α was set to 0.005 and Ω was set to 50. The training samples were generated on-line. The normalized mean squared error (*NMSE*) for every 2000 samples was collected during the learning.

Figure 5 shows the *NMSE*(dB) curve for the entire learning procedure. The target function and the SPSNN output are plotted in Figure 6(a) and (b), respectively. The SPSNN demonstrates good approximation.

Due to the quantization of the input variables, values falling into the same block are considered to be the same. To have a good approximation result, it is expected

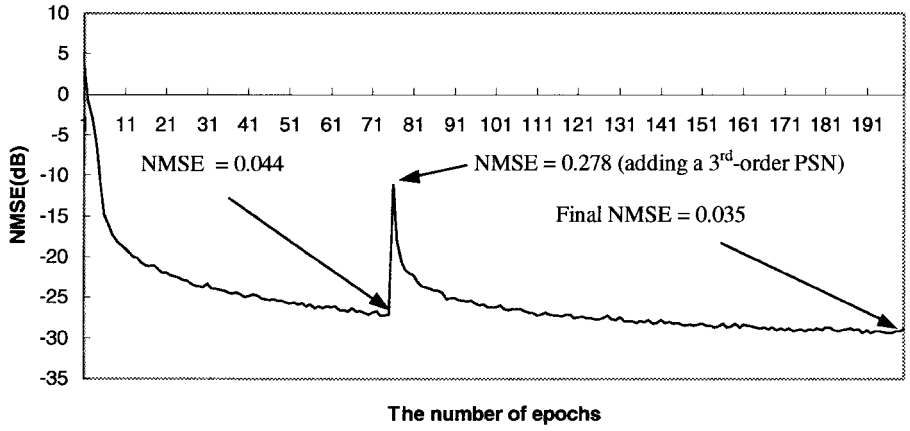


Figure 5. The learning curve for the approximation of Gabor function using the SPSNN.

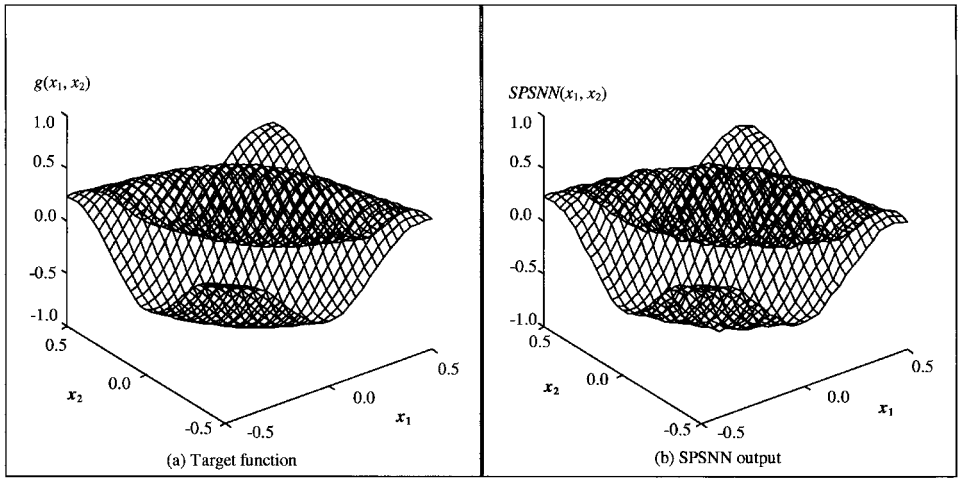


Figure 6. Plots for (a) The function g in Equation (10). (b) The output of the trained SPSNN.

that a finer quantization should be used in areas with much function variation. The following function is used for testing.

$$f(x_1, x_2, x_3, x_4) = (\ln(x_1x_2 + x_3x_4) + \ln(x_1x_3 + x_2x_4))^2, \quad 0.1 \leqslant x_1, x_2, x_3, x_4 \leqslant 3.1 \tag{11}$$

A fixed third-order SPSNN structure was used. Each input variable was quantized into 57 elements with 7 elements forming a block. The third-order SPSNN has 24 linear memory arrays and requires a memory size of 1512. The learning rate α was set to 0.02. The *NMSE* for every 15000 training patterns was collected during the training. After 25 epochs of training, the *NMSE* for the last 15000 training samples is 0.0306. To show the results, we plot the target function and the SPSNN output. Since there are four input variables, to plot we need to fix two of them. Figure 7(a) shows the plot for Equation (11) with x_2 and x_3 both set to 0.5. Figure 7(b) shows the plot for the approximated function by the SPSNN. Figure 7 shows large approximated errors in the input range [0.1, 1]. This is because that the output value of Equation (11) changes rapidly over this certain domain. To improve the approximation result, a finer quantization is required in this area.

4.2. PREDICTION

The Mackey-Glass (MG) Time Series [14, 17, 22] is 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:

$$y(k+1) = (1-b)y(k) + a\frac{y(k-\tau)}{1 + y^{10}(k-\tau)} \tag{12}$$

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

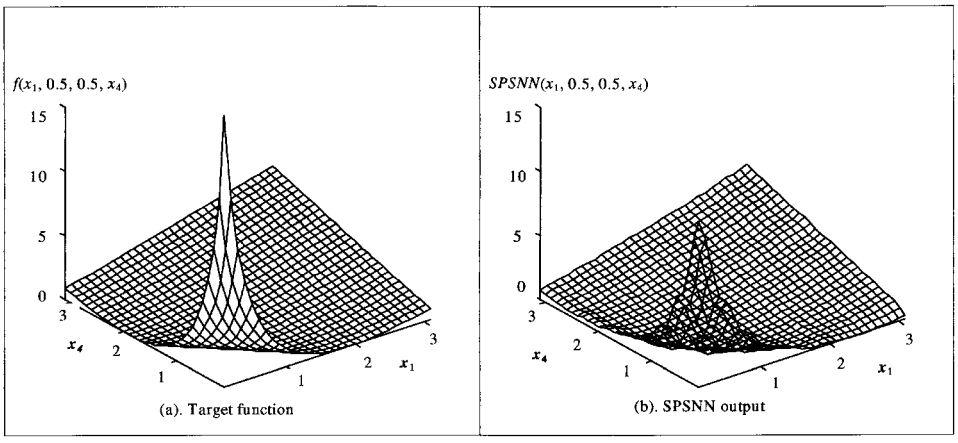


Figure 7. Plots for (a) The function f in Equation (11). (b) The output of the trained SPSNN.

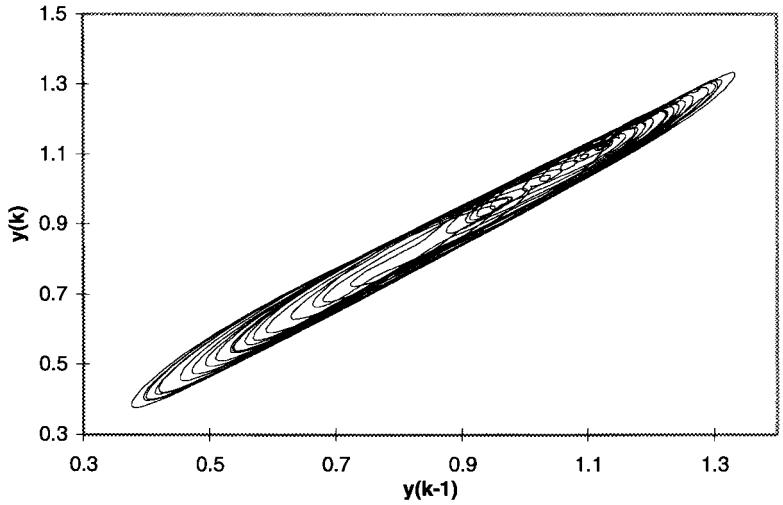
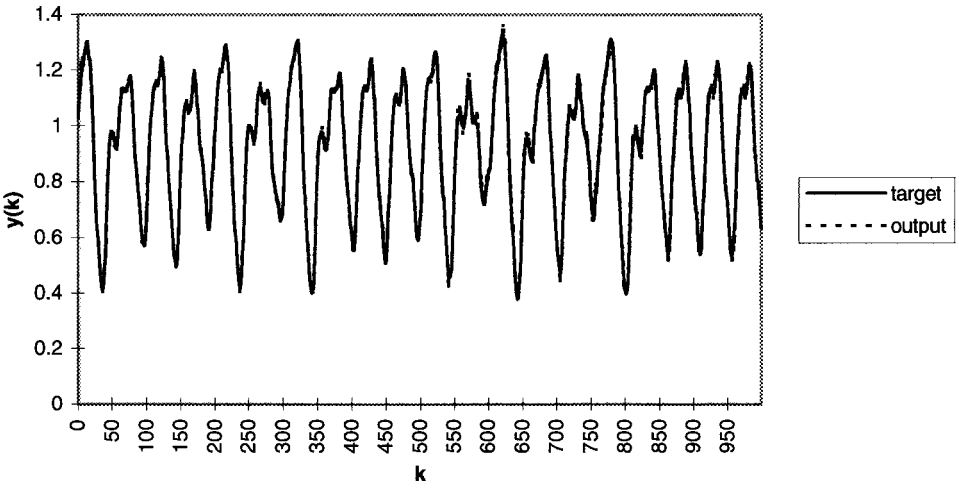


Figure 8. The state space diagram shows the quasiperiodic nature of the MG time series.

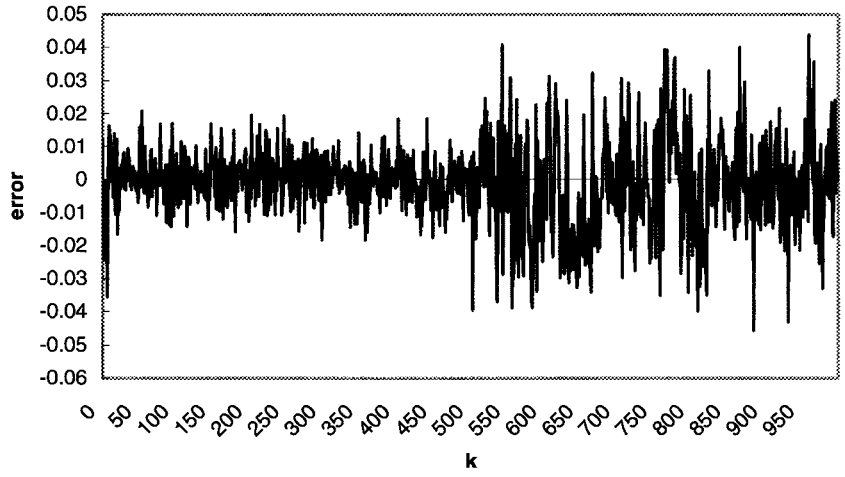
This model is complicated by the addition of a time delay τ in the nonlinear equation. The state space diagram in Figure 8 shows the quasiperiodic nature of the time series. The objective here is to model the time series and to predict the value of a time series at some future time, based on four previous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

Four measurements $y(\mathrm{k})$, $y(\mathrm{k}-6)$, $y(\mathrm{k}-12)$ and $y(\mathrm{k}-18)$ are used to predict $y(\mathrm{k}+1)$. A thousand data points were generated for the experiment. The first 500 data points were for training and the subsequent 500 data points were for testing. In this experiment, the SPSNN structure was allowed to grow during the training. Each input variable is quantized into 91 elements with 10 elements forming a block. The training started with a second-order SPSNN. The network was trained until the improvement on NMSE in two consecutive epochs became insignificant. A higher order PSN was then added. The learning rate α was set to 0.005 and Ω was set to 50%. The procedure continued until the *NMSE* for 500 training data dropped under 0.025 after 5000 epochs. The structure grew to an order of third to achieve this desired error level. This neural network required a memory size of 2400.

As described in the previous paragraph, 500 training data and 500 testing data had been generated for experimental use. Figure 9(a) shows these 1000 data (the solid line) and the prediction result from a trained SPSNN (the dash-line). The prediction is very accurate. While the two lines are very close, it is difficult to observe the dashed line. In the experiment, inputs to the SPSNN for making prediction are data generated from the Mackey-Glass difference equation. This emulates the use of *measured* data. We can call this one-step prediction; all information before and at the time instant k is used to predict the output value at time instant $k+1$. The prediction for the 500 testing data (data points 501 to 1000) is very accurate. Figure 9(b) shows the error.



(a)



(b)

Figure 9. (a) The result of prediction for the Mackey-Glass time series using the SPSNN. (b) The error of prediction.

The ability of long-term prediction has also been examined. For long-term prediction, the output of the SPSNN is fed back as the network inputs for calculating future values. Figure 10 shows the result for long-term prediction. It is noted that SPSNN was able to make good predictions until k equaled about 560 but it then began to fail.

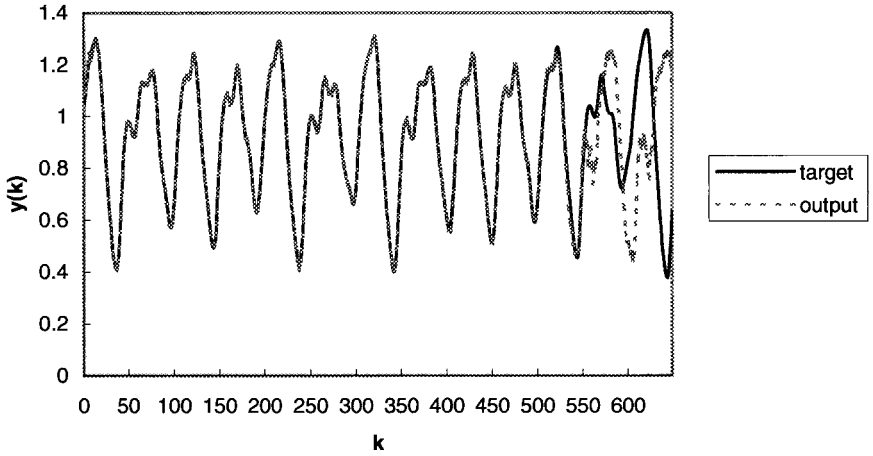


Figure 10. The long-term prediction result for the Mackey-Glass time series using the SPSNN. (prediction starts at $k = 501$).

The multilayer neural network has also been tested for a comparison. A three-layer feedforward neural network with 35 hidden nodes was used in this experiment. The learning rate α was set to 0.1. The network was trained using the error-backpropagation (BP) algorithm for 29000 epochs until the *NMSE* dropped to 0.05. Figure 11 shows the prediction results using 'measured' data. Figure 11(a) shows the exact function and the prediction curve and Figure 11(b) shows the error. Figure 12 shows that the multilayer neural network is unable to make long-term prediction using the feedback data from the neural network model.

4.3. CONTROL

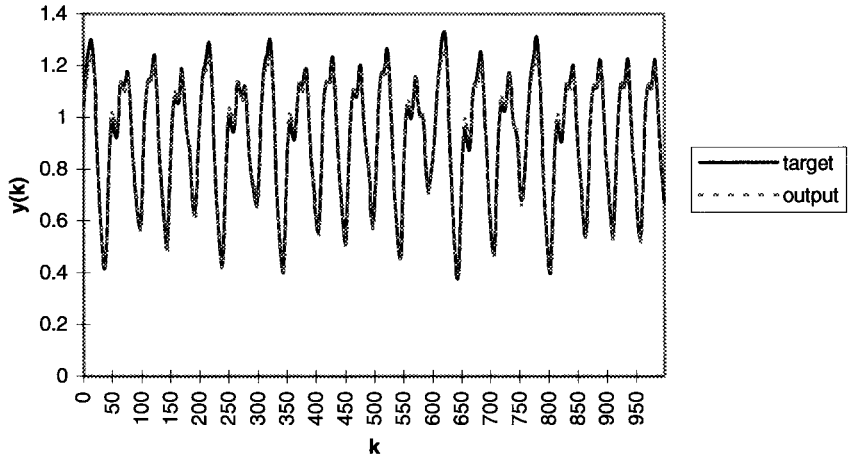
In this subsection, the SPSNN is applied to the identification and control of a nonlinear dynamic system. Figure 13 shows the model reference adaptive control (MRAC) structure used in this part of study.

In this MRAC technique, a reference model is first selected. The design of the controller is to make the plant output the same as the output of the reference model. In the structure in Figure 13, SPSNN_1 will learn to model the plant and SPSNN_2 will learn to control the plant to generate the same output as that from the reference model.

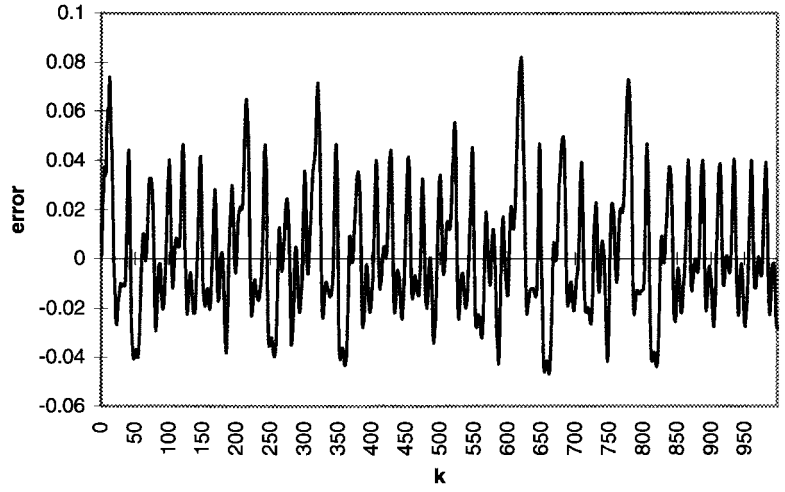
In the simulation, the plant is assumed to be a single-input and single-output system with unknown dynamics described by the following nonlinear difference equation

$$y(k+1) = g[y(k), y(k-1), y(k-2), u(k), u(k-1)] \tag{13}$$

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



(a)



(b)

Figure 11. (a) The result of prediction for the Mackey-Glass time series using the MNN. (b) The error of prediction.

$$g(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_2^2 + x_3^2} \tag{14}$$

This is one example used by Narendra and Parthasarathy [19]. The neural network structure SPSNN_1 used for identifying the plant is a third-order network. Each input variable is quantized into 91 elements with 10 elements forming a block. The SPSNN_1 was trained for 9 650000 time steps with the learning rate α equal to 0.0008 and a random input signal uniformly distributed in the interval $[-1,1]$.

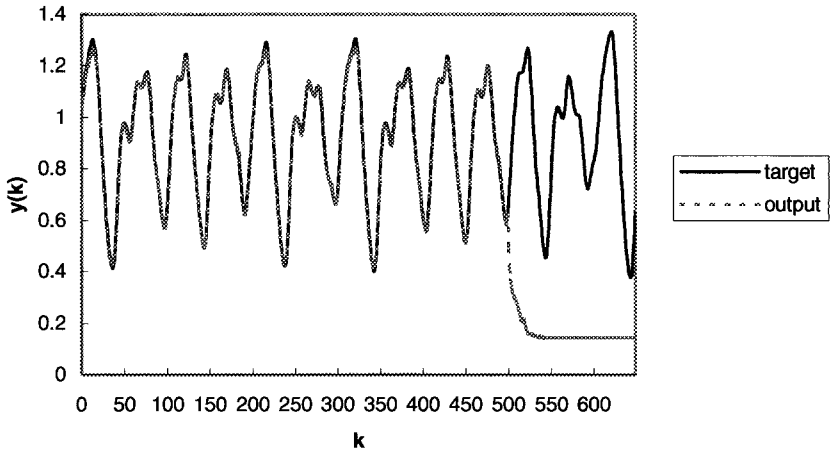


Figure 12. The long-term prediction result for the Mackey-Glass time series using the MNN. The well-trained MNN is unable to perform long-term prediction (prediction starts at $k = 501$).

On-line learning for developing the controller starts after a good plant model is generated. The controller is SPSNN_2, has y(k), y(k − 1), u(k − 1), and r(k) as inputs for generating the control input u(k). One may describe the function as

$$u(k) = \text{SPSNN}_2[y(k), y(k-1), u(k-1), r(k)]$$

The SPSNN_2 is also a third-order network. Each variable is quantized into 91 elements with 10 elements forming a block. The input $r(k)$ is selected to be

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

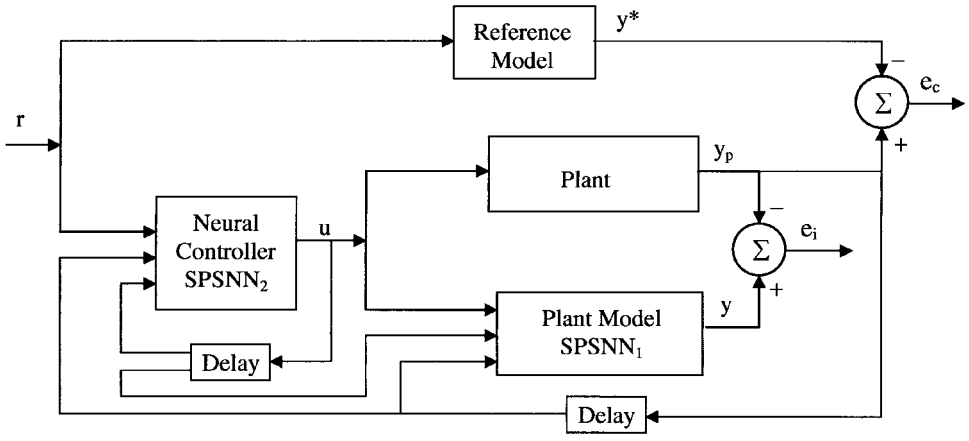


Figure 13. Model reference adaptive control with SPSNN.

Since $r(k)$ varies slowly with time, the reference model has been selected to be a single-period delay, i.e., $y^*(k+1) = r(k)$. $y^*(k+1)$ is the desired output in the MRAC technique. The controller was trained on-line with the learning rate α equal to 0.0000008. During the training, the weights of the neural controller $SPSNN_2$ were adjusted to reduce the control error $y^* - y_p$. The error was backpropagated through the plant model ($SPSNN_1$) into the controller ($SPSNN_2$) for weight adjustment. After 10 000000 time steps on-line training, the performance of the controller was tested. Figure 14 shows the error in control tracking.

4.4. CLASSIFICATION

The SPSNN has also been tested for classification, which the multilayer neural network is most suitable for. In this test, we create four classes using two 5-variable functions. These two functions are

$$g_1(x_1, x_2, x_3, x_4, x_5) = x_2 x_3 \exp[(x_3 x_4 - x_5)^2] - 2(x_1 x_4 - 1)^2 x_5 + x_3 - x_4 - 0.4 \tag{15}$$

$$g_2(x_1, x_2, x_3, x_4, x_5) = x_2 x_3 (2x_4 x_5 - 1) - \sin(1.5\pi x_1) - 1.0 \tag{16}$$

where $0 \leqslant x_1, x_2, x_3, x_4, x_5 \leqslant 2.0$.
$g_1(x_1, x_2, x_3, x_4, x_5) = 0$ and $g_2(x_1, x_2, x_3, x_4, x_5) = 0$ are used as boundaries for four classes:

class I:
$g_1 \geqslant 0$ and $g_2 \geqslant 0$
class II:
$g_1 \geqslant 0$ and $g_2 < 0$

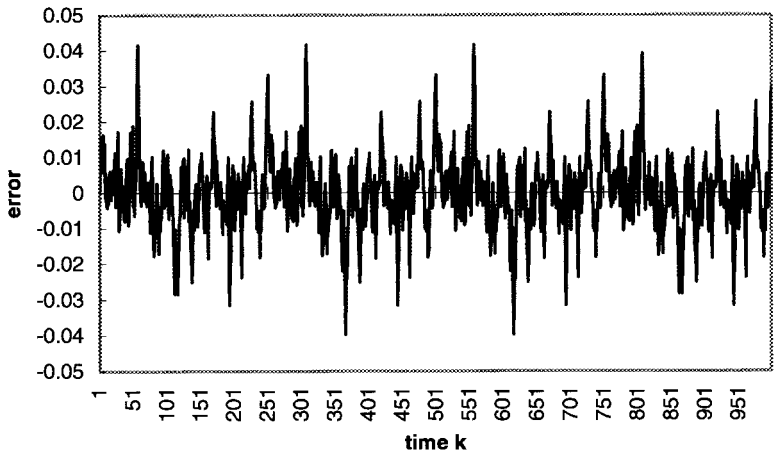


Figure 14. Output errors with $SPSNN_2$ as the controller (control output range is in $[-0.6, 0.6]$).

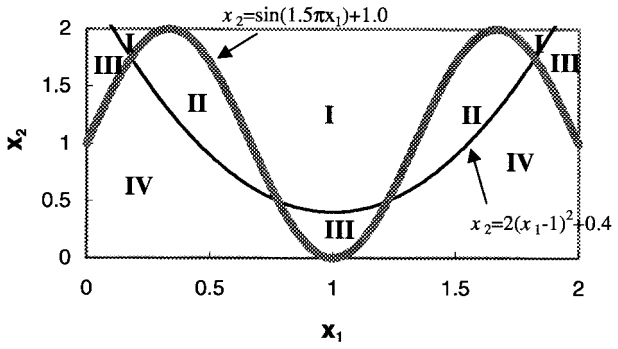


Figure 15. The distribution of four classes with $x_3 = x_4 = x_5 = 1.0$.

class III:

$g_1 < 0$ and $g_2 \geqslant 0$

class IV:

$g_1 < 0$ and $g_2 < 0$

Figure 15 shows the distribution of four classes on the x1 ~ x2 plane with the other three variables set to 1.0.

A fixed third-order SPSNN structure was used for learning the classification. Each input variable is quantized into 82 elements with each block consists of 10 elements. Thirty thousand training samples were generated evenly within the input domain. The learning rate α was selected to be 0.001. After 2000 epochs of training, 3000 patterns were tested. The SPSNN correctly classified 94.1% of test patterns. Note that while the patterns are randomly generated in the input domain, some will be very close to the boundary and be 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 network was trained using the error backpropagation (BP) algorithm for 5000 epochs and the learning rate was set to 0.1. The MNN correctly classified 93.1% of test patterns. The accuracy is about the same as that for SPSNN but the learning time is much longer.

5. Conclusion

A sigma-pi-sigma neural network has been developed and tested. The novel structure is a memory-based neural network that can self-generate the necessary basis functions. While 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 higher order PSNs during the learning. This makes the guess of the number of the order not

necessary. The SPSNN combines the table lookup techniques (such as CMAC) and the computation-based techniques (such as MNN and RBFN). A table lookup technique heavily relies on data memorization and requires very little computation. The SPSNN overcomes the huge memory size problem, which exists in the conventional CMAC [1, 2, 7, 10, 21] in high-dimensional modeling. Our experiments show that learning usually converges easily. This is an important merit of the SPSNN compared to the multilayer neural networks.

The 'product' and 'sum' operators endow the three new structures fitting capabilities. Actually, there is a nature neurobiological interpretation for this type combination of product and sum operations. Local regions of dendritic arbor could act as product units whose outputs are summed at the soma [8]. In neurophysiology, the possibility that dendritic computations could include local multiplicative nonlinearities is widely accepted. Mel and Koch [18] argued that sigma-pi units underlie the learning of nonlinear associative maps in cerebral cortex. The discussions above make us believe that this approach could lead us to develop a new computational model that is biologically plausible and more powerful than the currently used neural networks.

Further researches on theoretical analysis and improvement of the SPSNN are possible. One possible improvement on the learning technique is the adaptive input quantization. The input quantization may be made adjustable. A finer quantization should be applied in areas with much function variation, and a rougher quantization in flatter areas. One example is on the approximation of the Equation (11). This could further improve the approximation accuracy and reduce the memory size. With the use of overlapped rectangular pulses as basis functions, the output of an SPSNN is piecewise constant. This creates inconveniences in applications that require derivatives of the model output with respect to inputs. The use of continuous basis functions (such as Gaussian function and B-spline function) will help generate a differentiable network output.

References

1. Albus, J. S.: A new approach to manipulator control: The Cerebellar Model Articulation Controller (CMAC), *Journal of Dynamic Systems, Measurement, and Control, Transaction of ASME*, (1975a, Sept.), 220–227.
2. Albus, J. S.: Data storage in the cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control, Transaction of ASME*, (1975b, Sept), 228–233.
3. Bianchini, M., Fransconi, P. and Gori, M.: Learning without local minima in radial basis function networks, *IEEE Transactions on Neural Networks*, **6** (1995), 749–756.
4. Chen, S., Mulgrew, B. and Grant, P. M.: A clustering technique for digital communications channel equalization using radial basis function networks, *IEEE Transactions on Neural Networks*, **4** (1993), 570–579.
5. Chen, T. and Chen, H.: Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural nertworks, *IEEE Transactions on Neural Networks*, **6** (1995), 904–910.

6. Cheng, Y. H. and Lin, C. S.: A learning algorithm for radial basis function networks: with capability of adding and pruning neurons, *Proceedings of IEEE International Conference on Neural Networks*, (1994).
7. Chiang, C. T. and Lin, C. S.: CMAC with general basis functions, *Neural Networks*, **9** (1996), 1199–1211.
8. Durbin, R. and Rumelhart, D. E.: Product units: A computationally powerful and biologically plausible extension to backpropagation networks, *Neural Computation*, **1** (1989), 133–142.
9. Elanayar, Sunil V. T. and Shin, Y. C.: Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems, *IEEE Transactions on Neural Networks*, **5** (1994), 594–603.
10. Glanz, F. H., Miller, W. T. and Kraft, L. G.: An overview of the CMAC neural network, *Proceedings of the IEEE Conference on Neural Networks for Ocean Engineering, Washington DC*, (1991, Aug.), 301–308.
11. Gorinevsky, D.: On the persistency of excitation in radial basis function network identification of nonlinear systems, *IEEE Transactions on Neural Networks*, **6** (1995), 1237–1244.
12. Hecht-Nielsen, R.: Theory of the backpropagation neural network, *Proceedings of International Joint Conference on Neural Networks*, **1** (1989), 593–611.
13. Hornik, K., Stinchcombe, M. and White, H.: Multi-layer feedforward networks are universal approximators, *Neural Networks*, **2** (1989), 359–366.
14. Lapedes and Farber, R.: Nonlinear signal processing using neural networks: Prediction and system modeling, *Technical Report LA-UR-87-2662*, Los Alamos National Laboratory, Los Alamos, NM.
15. Lee, S. and Kil, R. M.: A gaussian potential function network with hierarchically self-organizing learning, *Neural Networks*, **4** (1991), 207–224.
16. Leonard, J. A., Kramer, M. A. and Ungar, L. H.: Using radial basis functions to approximate a function and its error bounds, *IEEE Transactions on Neural Networks*, **3** (1992), 624–627.
17. Mackey, M. and Glass, L.: Oscillation and chaos in physiological control systems, *Science*, **197** (1977), 287.
18. Mel, B. W. and Koch, C.: Sigma-pi learning: on radial basis functions and cortical associative learning, In: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, Morgan-Kaufmann, San Mateo, CA pp. 474–481, 1990.
19. Narendra, K. S. and Parthasarathy, K.: Identification and control of dynamical systems using neural networks, *IEEE Transactions on Neural Networks*, **1** (1990), 4–27.
20. Nedeljkovic, V.: A novel multilayer neural networks training algorithm that minimizes the probability of classification error, *IEEE Transactions on Neural Networks*, **4** (1993), 650–659.
21. Parks, P. C. and Militzer, J.: Comparison of five algorithms for the training of CMAC memories for learning control systems, *Automatica*, **28** (1992), 1027–1035.
22. Platt, N.: A resource-allocating network for function interpolation, *Neural Computation*, **3** (1991), 213–225.
23. Rumelhart, D. E., Hinton, G. E. and Williams, R. J.: Learning Internal Representation by Error Propagation, In: D. E. Rumelhart, and J. L. McClelland (eds), *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, MIT Press, Cambridge, MA: **1** (pp.318–362).
24. Sangers, T. D.: A tree-structure adaptive network for function approximation in high-dimensional space, *IEEE Transactions on Neural Networks*, **2** (1991), 285–293.

25. Shin, Y. and Ghosh, J.: Ridge polynomial networks, *IEEE Trans. Neural Networks*, **6** (1995), 610–622.
26. Werbos, P. J.: Beyond Regression: New Tools for Predicting and Analysis in the Behavioral Sciences, Ph.D. dissertation, Harvard University: Boston, MA, 1974.
27. Zhang, Q. and Benveniste, A.: Wavelet network, *IEEE Transactions on Neural Network*, **3** (1992), 889–898.
28. Zhang, J., Walter, G. G., Miao, Y. and Lee, Wan Ngai Wayne.: Wavelet neural networks for function learning, *IEEE Transactions on Signal Processing*, **43** (1995), 1485–1497.