1998 International Computer Symposium
Workshop on Artificial Intelligence
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A MEMORY-BASED SIGMA-PI-SIGMA NEURAL NETWORK (SPSNN)

Chien-Kuo Li,, and Jung-Hung Hung
Department of Information Management
Shih Chien University, Taipei, Taiwan, R.O.C.
Email: ckli@scc1.scc.edu.tw

Abstract

This paper presents a class of self-organized basis function networks called sigma-pi-sigma neural networks (SPSNNs). A neural network structure that uses small memory blocks plus additional operators as basic building blocks has been investigated. The output of the new structure is the sum of the outputs from several submodules. Each submodule consists of memory-based product-of-sum form neural networks. The memory contents in these submodules are adjusted during the learning process. The new structure can learn to implement static mapping that multilayer neural networks and radial basis function networks usually do. The new neural network structure demonstrates excellent learning convergence characteristics and requires small memory space.

I. Introduction

This study investigated a neural network structure that uses small memory blocks plus additional operators as basic building blocks. While typical basis function networks select a specific type of basis function such as the Gaussian function [1] for function approximation and modeling, the new structure intends to use "self-organized" basis functions. These basis functions are generated during the learning process. Memory blocks together with additional operators are especially suitable for implementing such basis functions. Each memory block can implement a function for a subset of inputs. Operators, usually arithmetic ones, are applied to these functions in order to approximate high-dimensional complicated functions or to model systems.

The new structure overcomes major problems encountered in multilayer neural networks (MNNs) [2, 3, 4] and radial basis function networks (RBFNs) [1, 5, 6, 7, 8], which are widely used for static mapping. It is known that MNNs often encounter difficulty in learning. For a slightly complicated mapping, it is hard to predict how long the learning will take and whether the learning will converge to an acceptable result. Another type of neural networks, RBFNs, often uses the Gaussian function as the basis function. Since a Gaussian function provides fitting in a local area, the learning convergence is a less difficult problem compared to that in MNNs. 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 more input variables. To increase the fitting power of each basis function in order to reduce the needed basis functions, many researchers suggested to make the Gaussian function scaleable in each dimension and rotatable in the input space [1, 7, 8]. The trade-off is the increased learning difficulty.

The new structure is a sigma-pi-sigma neural network (SPSNN), which consists of product-of-sum networks. The SPSNN is similar to ridge polynomial networks (RPNs) [9], which use a special form of ridge polynomials. While, in RPNs, the pi-sigma networks use polynomial terms, the SPSNN uses linear memory arrays that self-generated 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.

II. The New Neural Network Structure

A.The Structure

Figure 1 shows the memory-based pi-sigma network having a K-th order pi-sigma network structure. Its output is the product of the outputs from K submodules. The network output P_K has the "product-of-sum" form $\prod_{i=1}^{K}\sum_{j=1}^{N_v} f_{Kij}(x_j)$, where x_j's are inputs, N_v is the number of inputs, $f_{Kij}()$ is a function generated through network training, and K is the order of the submodules. Figure 2 shows a SPSNN whose output is the sum of outputs from K different orders of pi-sigma networks. The output $SPSNN(\cdot)$ equals $\sum_{n=1}^{K}\prod_{i=1}^{n}\sum_{j=1}^{N_v} f_{nij}(x_j)$.

The output of a (linear) memory array, denoted by $f_{ij}(x_j)$, is calculated as $\sum_{k=1}^{N_q+N_e-1} w_{ijk}B_{jk}(x_j)$, where x_j's are inputs, $B_{jk}()$ is a single-variable basis function, w_{ijk}'s are weight values stored in the memory, N_q and N_e are two selected constants. Further discussion on N_q and N_e will be given later. Although $B_{jk}()$ can be any adequate basis function, in this section, focus will be 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 the simple addition of retrieved w_{ijk}'s. Since only w_{ijk}'s need to be memorized, a linear memory array is enough for this purpose

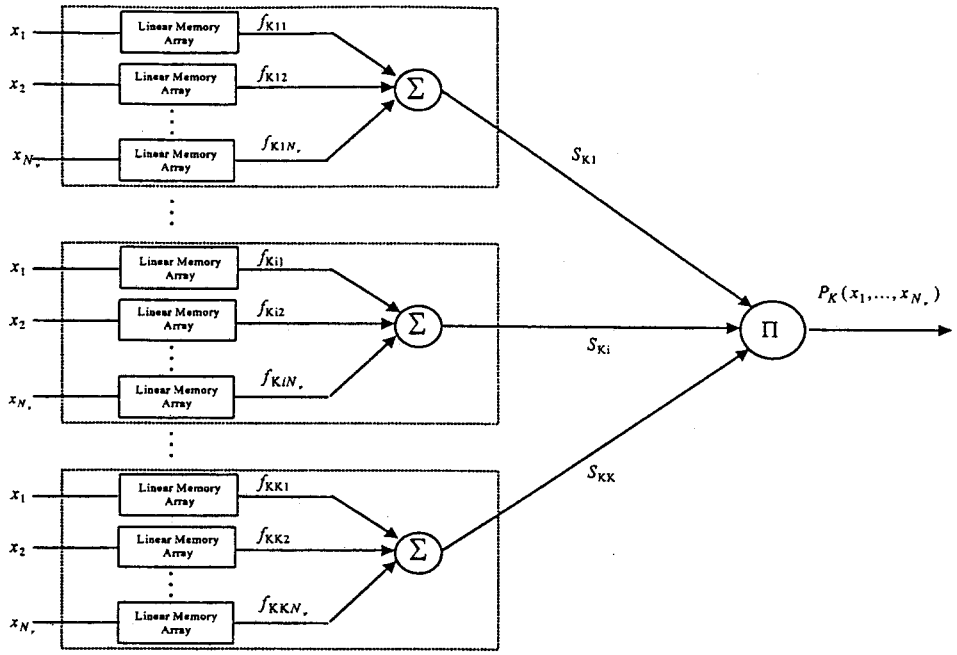


Figure 1. A basic building block (K-th order PSN) for the SPSNN.

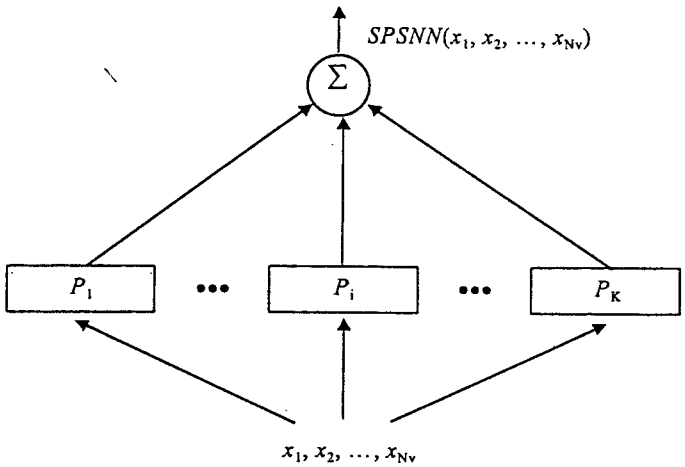
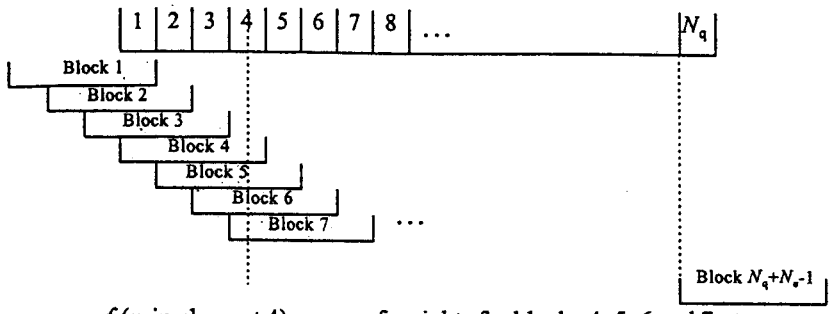


Figure 2. A K-th order SPSNN is composed of K different orders of PSNs; P_i represents an i-th order PSN as shown in Figure 1.

Figure 3 illustrates the arrangement of the overlapped rectangular pulse functions, and the computation of f_{ij} with an given input x_j. The range of each input variable (e.g., x_j) is divided into N_q elements and every N_e neighboring elements are grouped into a block. Each block is assigned a basis function, which may be a bell-shape function, a cubic spline function, a triangular or a reetangular pulse function. In this study, we select to focus on the rectangular pulse function. With such a basis function, each element is covered by N_e blocks, the computation of $f_{ij}(x_{sj})$ is simply the addition of weights associated to the N_e blocks covering the specific x_{sj}. The arrangement makes that learning for one element also alters the values in the neighborhood. This creates 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 N_p submodules and N_v input variables, the total memory size required will be $N_p \times N_v \times (N_q+N_e-1)$. A memory size of 20k, which is considered small, is required for a structure with 20 submodules, 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.

While the SPSNN uses single-variable memory arrays, for a K-th order SPSNN, the required memory size is $\frac{K(K+1)}{2} N_v (N_q + N_e - 1)$.



$f_{ij}(x_j$ in element 4) = sum of weights for blocks 4, 5, 6 and 7

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

B. Learning Algorithms

Learning algorithms for the SPSNN are based on gradient descent on the mean squared error surface in weight space. The mean squared error is given as

$$E = \sum_{s=1}^{L} E_s = \frac{1}{L}\sum_{s=1}^{L}(t_s - SPSNN_s)^2 \quad (1)$$

where E_s represents the error on pattern s, t_s is the desired (target) output for pattern s, L is the number of samples, and $SPSNN_s$ is the actual output of the SPSNN when pattern s is presented. The objective of learning is to find a set of weights that minimizes the function E. The following equations can be used to derive the learning rule for the K-th order SPSNN:

$$SPSNN(\cdot) = P_1 + P_2 + \ldots + P_n + \ldots + P_K \quad (2)$$

$$P_n = \prod_{i=1}^{n} S_{ni} \quad (3)$$

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

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

The gradient descent method makes a change on a weight proportional to the negative derivative of the mean squared error with respect to the weight. Thus the learning rule is

$$\Delta w_{nijk}\Big|_{x_s} \propto -\frac{\partial E_s}{\partial w_{nijk}} = -\frac{\partial E_s}{\partial P_n}\frac{\partial P_n}{\partial S_{ni}}\frac{\partial S_{ni}}{\partial f_{nij}}\frac{\partial f_{nij}}{\partial w_{nijk}} \quad (6)$$

and the amount to be updated for w_{nijk} is

$$\Delta w_{nijk}\Big|_{x_s} = \frac{\alpha}{K}(t_s - SPSNN_s)\{\prod_{p\neq i} S_{np}\}B_{jk}(x_{sj}) \quad (7)$$

where α is a learning rate. With the rectangular pulse basis function, only N_e $B_{jk}(x_{sj})$'s have nonzero values. Thus updating will occur only for those N_e corresponding memory elements (*i.e.*, weights).

C. Learning Procedure

Fixed structure

The following summarizes a basic learning procedure in which the order of the SPSNN is determined before the training starts:

1. Initialize all memory arrays with random memory contents between -δ and δ.
2. Obtain a training sample.
3. Compute the overall output for this sample and calculate the error.
4. Update all memory arrays using (7).
5. If the error for the past N samples is acceptable, then stop.

   Otherwise, go to step 2.

Flexible structure

The following summarizes a learning procedure in which in which the network is allowed to grow to an adequate order:

1. Initialize the neural network with the Kth-order SPSNN. Select all learning parameters.
2. Initialize all memory arrays of the newly added pi-sigma submodule (with order > K) with random memory contents between -δ and δ.
3. Obtain a training sample.
4. Compute the overall output for this sample and calculate the error.
5. Use Ω percent of the error to update the new pi-sigma submodule and $(100\text{-}\Omega)/\hat{K}$ percent of the error for each of the old pi-sigma submodules, where $\hat{K}$ is the current order of the old network.
6. Update all memory arrays using (7).
7. If the error for the past N_l samples is acceptable, then stop.

8. If the improvement in the last N_2 samples is insignificant, then add one more pi-sigma network (with an order higher than the current one by one) to the neural network and go to step 2. Otherwise, go to step 3.

III. An Example and Results

In this section, results for approximating a 2-D Gabor function [9] is provided for an illustration of the new SPSNN technique. The convolution version of complex 2-D Gabor functions has the following form

$$g(x_1, x_2) = \frac{1}{2\pi\lambda\sigma^2} e^{-\{[(x_1/\lambda)^2 + x_2^2]/2\sigma^2\}} e^{2\pi i(u_0 x_1 + v_0 x_2)} \quad (8)$$

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.

$$g(x_1, x_2) = \frac{1}{2\pi(0.5)^2} e^{-[(x_1^2 + x_2^2)/2(0.5)^2]} \cos(2\pi(x_1 + x_2)) \quad (9)$$

The normalized mean squared error (*NMSE*) as a measure in the learning process is defined as

$$NMSE = \frac{\sqrt{MSE}}{\sqrt{\frac{\sum_{s=1}^{n}(y_{ts} - \bar{y})^2}{n}}} \quad (10)$$

where *MSE* is the mean squared error, y_{ts} is the target output value for sample input s, and $\bar{y}$ is the mean value of the target outputs.

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 (9). 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 1,728. 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 2,000 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 Figure 6(b), respectively. The SPSNN demonstrates good approximation. The outputs of three PSNs are also provided in Figures 7(a), (b) and (c).

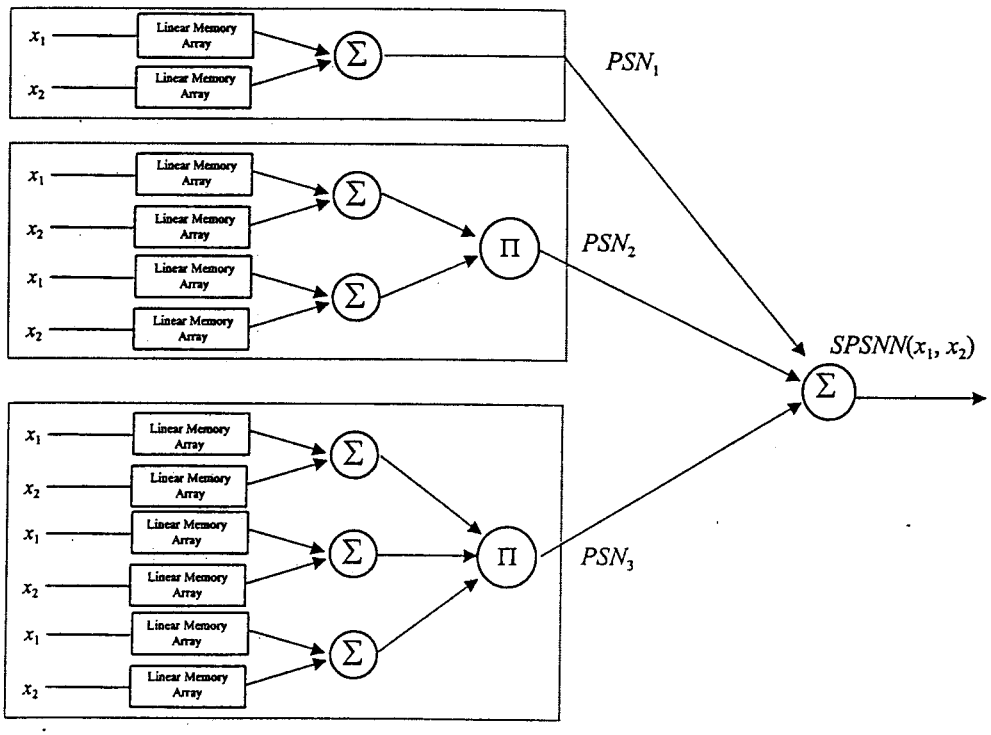


Figure 4. A third-order SPSNN used for approximating a Gabor function (9), where PSN_n is an n-*th* order PSN.

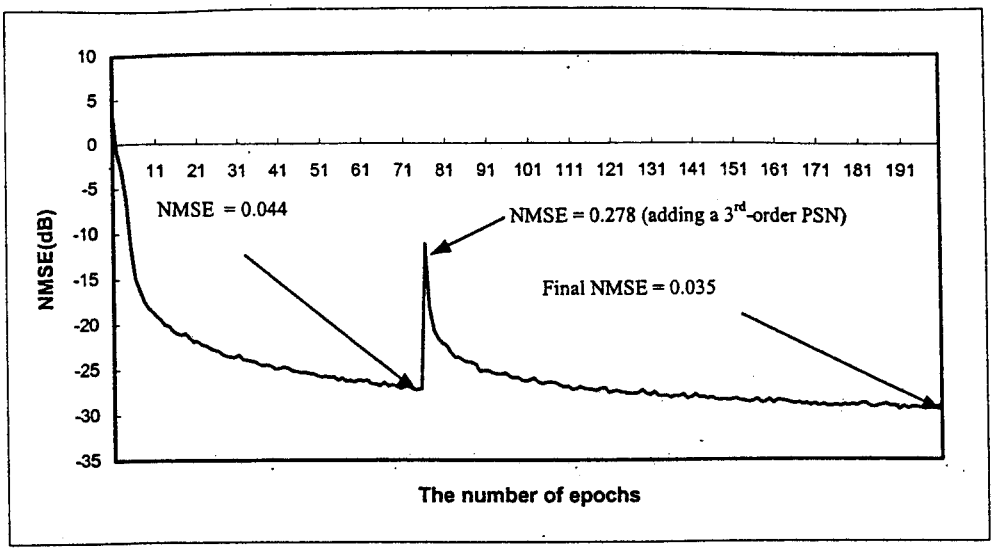


Figure 5. The learning curve for the approximation of Gabor function using the SPSNN.

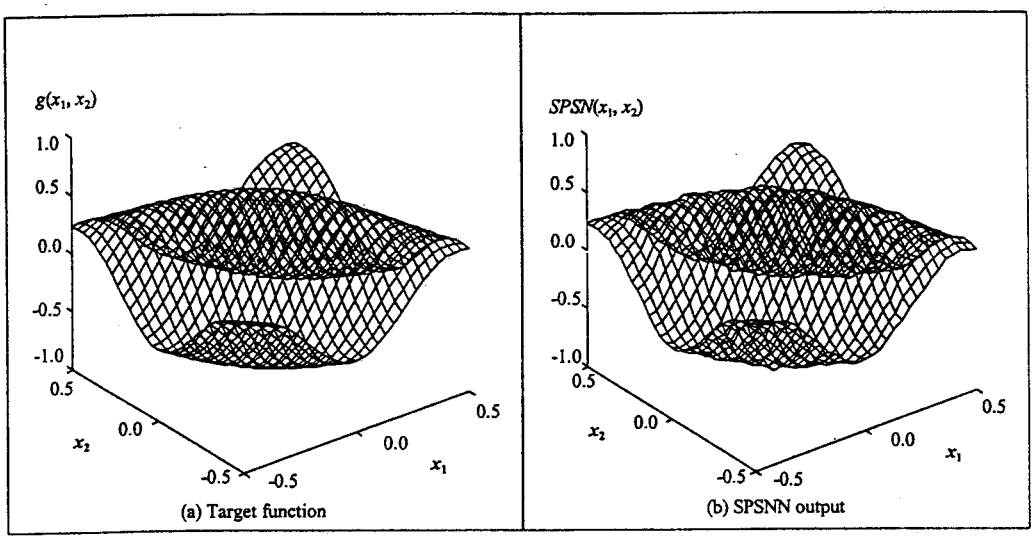


Figure 6. the target function g in (9) and the network output.

IV. 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 new submodules during the learning. This makes the guess of the number of submodules not necessary.

The "product" and "sum" operators endow the new structure 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 [10]. In neurophysiology, the possibility that dendritic computations could include local multiplicative nonlinearities is widely accepted. Mel and Koch [11] argue that sigma-pi units underlie the learning of nonlinear associative maps in the 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.

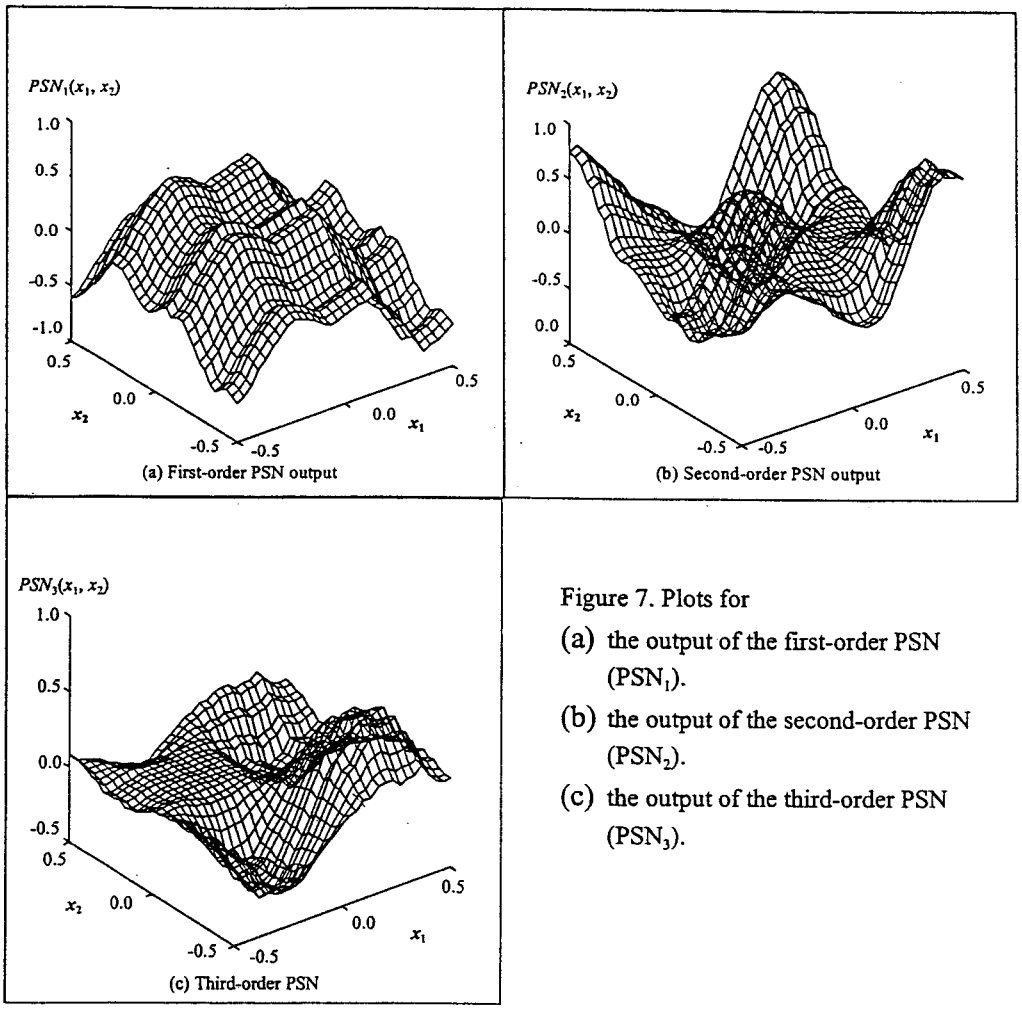


Figure 7. Plots for

(a) the output of the first-order PSN (PSN_1).

(b) the output of the second-order PSN (PSN_2).

(c) the output of the third-order PSN (PSN_3).

References

1. Lee, S. & Kil, R. M., " A Gaussian Potential Function Network with Hierarchically Self-Organizing Learning," Neural Networks, vol. 4, pp. 207-224, 1991.

2. Rumelhart, D. E., Hinton, G. E. & Williams, R. J., "Learning Internal Representation by Error Propagation." in Rumelhart, D. E. & McClelland, J. L. (Eds), Parallel Distributed Processing: Exploration in the Microstructure of Cognition, vol. 1 (pp. 318-362). Cambridge, MA: MIT Press, 1986.

3. Hecht-Nielsen, R., "Theory of the Backpropagation Neural Network," Proceedings of International Joint Conference on Neural Networks, vol. 1, pp. 593-611, 1989

4. Hornik, K., Stinchcombe, M. & White, H., "Multi-layer Feedforward Networks are Universal Approximators," Neural Networks, vol. 2, pp. 359-366, 1989

5. Zhang, J., Walter, G. G., Miao, Y. & Lee, Wan Ngai Wayne, "Wavelet Neural Networks for Function Learning'" IEEE Transactions on Signal Processing, vol. 43, pp. 1485-1497, 1995.

6. Cheng, Y. H. & 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. Chen, T. & Chen, H., "Approximation Capability to Functions of Several Variables, Nonlinear Functionals, and Operators by Radial Basis Function Neural Nertworks," IEEE Transactions on Neural Networks, vol. 6, pp. 904-910, 1995.

8. Zhang, Q. & Benveniste, A., " Wavelet Network," IEEE Transactions on Neural Network, vol. 3, pp. 889-898, 1992.

9. Shin, Y. and Ghosh, J., "Ridge Polynomial Networks," IEEE Trans. Neural Networks, vol. 6, no. 3, pp. 610-622, May 1995.

10. Durbin, R. and Rumelhart, D. E., "Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks," neural Computation, vol. 1, pp. 133-142,1989.

11. Mel, B. W. and Koch, C., "Sigma-Pi Learning: on Radial Basis Functions and Cortical Associative Learning," in Advances in Neural Information Processing Systems 2, D. S. Touretzky, Ed. San Metro, CA: Morgan-Kaufmann, pp. 474-481, 1990