Since Eq. 27 is a special case of Eq. 28, this completes the proof. □

B Generalized Weierstrass Polynomial Approximation Theorem

The well-known Weierstrass polynomial approximation theorem for univariate continuous functions can be generalized to multi-dimensional settings. This generalized theorem specifies the uniform approximation capability of polynomials.

Theorem 3 (Generalized Weierstrass polynomial approximation theorem [7]) *Let $K \subset \Re^d$ $(d \geq 1)$ be compact. Then for any function $f \in C(K)$, the vector space of continuous scalar functions on K, there exists a polynomial $p \in C(K)$ such that for given $\epsilon > 0$, $|f(\mathbf{x}) - p(\mathbf{x})| < \epsilon$ for all $\mathbf{x} \in K$.*

For hypercubes $[0,1]^d$, the generalized multivariate Bernstein polynomials [7] are shown to have the property mentioned in the theorem.

The theorem stated above can be obtained as a corollary of the Stone-Weierstrass theorem:

Theorem 4 (Stone-Weierstrass theorem [31]) *Let $K \subset \Re^d$ $(d \geq 1)$ be compact, and A be an algebra of continuous scalar functions on K which separates the points in K and which contains the constant functions. Then A is dense in $C(K)$, the vector space of continuous scalar functions on K.*

A subspace A of $C(K)$ is an algebra if whenever $f, g \in A$, then $f \cdot g \in A$. An algebra A separates the points in K if for any $\mathbf{x}, \mathbf{y} \in K$ and $\mathbf{x} \neq \mathbf{y}$, there exists

a function $f \in A$ such that $f(\mathbf{x}) \neq f(\mathbf{y})$. The generalized Weierstrass polynomial approximation theorem follows from this theorem since the family of polynomials is an algebra which satisfies all the conditions of the theorem.

A Proof of Theorem 2

Proof: We want to show that right-hand-side of Eq. 7 can be expressed as a special case of right-hand-side of Eq. 8. From Eq. 7,

$$\begin{aligned}\sum_{j=0}^{k}\sum_{m=1}^{n_j} a_{jm} <\mathbf{x},\mathbf{v}_{jm}>^j &= \sum_{m=1}^{n_0} a_{0m} + \sum_{m=1}^{n_1} a_{1m} <\mathbf{x},\mathbf{v}_{1m}> + \cdots + \sum_{m=1}^{n_k} a_{km} <\mathbf{x},\mathbf{v}_{km}>^k \\ &= \sum_{l=0}^{k} p_l(\mathbf{x}), \end{aligned} \tag{24}$$

where

$$p_0(\mathbf{x}) = \sum_{m=1}^{n_0} (<\mathbf{x},\mathbf{0}> + a_{0m})(<\mathbf{x},\mathbf{0}> +1)^{m-1}, \tag{25}$$

$$p_l(\mathbf{x}) = \sum_{m=1}^{n_l} (<\mathbf{x}, a_{lm}\mathbf{v}_{lm}> +0)(<\mathbf{x},\mathbf{v}_{lm}> +0)^{l-1}(<\mathbf{x},\mathbf{0}> +1)^{n_0+\cdots+n_{l-1}+m-l}, \quad l = 1,\cdots,k, \tag{26}$$

$\mathbf{x},\mathbf{v}_{jm} \in \Re^d$, $a_{jm} \in \Re$, $a_{jm}\mathbf{v}_{jm} = (a_{jm}v_{jm1},\cdots,a_{jm}v_{jmd}) \in \Re^d$, and $\mathbf{0} = (0,\cdots,0) \in \Re^d$.

By proper change of variables ($j = m$ for $p_0(\mathbf{x})$ and $j = n_0+\cdots+n_{l-1}+m$ for $p_l(\mathbf{x})$, $l = 1,\cdots,k$), Eq. 24 becomes

$$\begin{aligned}\sum_{j=0}^{k}\sum_{m=1}^{n_j} a_{jm} <\mathbf{x},\mathbf{v}_{jm}>^j &= \sum_{l=0}^{k} p_l(\mathbf{x}) \\ &= \sum_{j=1}^{n_0}(<\mathbf{x},\mathbf{0}> + a_{0j})(<\mathbf{x},\mathbf{0}> +1)^{j-1} \\ &\quad + \sum_{j=n_0+1}^{n_0+n_1} (<\mathbf{x}, a_{1,j-n_0}\mathbf{v}_{1,j-n_0}> +0)(<\mathbf{x},\mathbf{0}> +1)^{j-1} \\ &\quad \vdots \\ &\quad + \sum_{j=n_0+\cdots+n_{k-1}+1}^{n_0+\cdots+n_k} (<\mathbf{x}, a_{k,j-(n_0+\cdots+n_{k-1})}\mathbf{v}_{k,j-(n_0+\cdots+n_{k-1})}> +0) \times \\ &\quad (<\mathbf{x},\mathbf{v}_{k,j-(n_0+\cdots+n_{k-1})}> +0)^{k-1}(<\mathbf{x},\mathbf{0}> +1)^{j-k}. \end{aligned} \tag{27}$$

Also from Eq. 8,

$$\begin{aligned}\sum_{j=1}^{N}\prod_{i=1}^{j}(<\mathbf{x},\mathbf{w}_{ji}> + w_{ji}) &= \sum_{j=1}^{n_0}\prod_{i=1}^{j}(<\mathbf{x},\mathbf{w}_{ji}> + w_{ji}) + \sum_{j=n_0+1}^{n_0+n_1}\prod_{i=1}^{j}(<\mathbf{x},\mathbf{w}_{ji}> + w_{ji}) + \cdots \\ &\quad + \sum_{j=n_0+\cdots+n_{k-1}+1}^{n_0+\cdots+n_k}\prod_{i=1}^{j}(<\mathbf{x},\mathbf{w}_{ji}> + w_{ji}). \end{aligned} \tag{28}$$

[38] Y. Shin, J. Ghosh and D. Samani, "Computationally efficient invariant pattern classification with higher-order pi-sigma networks," in *Intelligent Engineering Systems through Artificial Neural Networks*, C. H. Dagli, L. I. Burke and Y. C. Shin (Eds.), ASME Press, Vol.2, pp.379-384, 1992.

[39] D. F. Specht, "Generation of polynomial discriminant functions for pattern recognition," *IEEE Trans. on Electron. Comput.*, Vol.EC-16, pp.308-319, 1967.

[40] M. F. Tenorio and W. Lee, "Self-Organizing network for optimum supervised learning," *IEEE Trans. on Neural Networks*, Vol.1, No.1, pp.100-110, Mar. 1990.

[41] K. Tumer, "Structural adaptation and generalization in neural networks," M. S. Thesis, The University of Texas at Austin, Aug. 1992.

[42] S. S. Venkatesh, "Programmed interactions in higher-order neural networks: maximal capacity," *Journal of Complexity*, Vol.7, pp.316-337, 1991.

[43] M. Vidyasagar, *Nonlinear Systems Analysis*, (2nd Ed.), Prentice-Hall, 1993.

[44] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, 1985.

[45] Y. Zhao and C. G. Atkeson, "Some approximation properties of projection pursuit learning networks," in *Advances in Neural Information Processing Systems 4*, S. J. Hanson, J. E. Moody and R. P. Lippmann (Eds.), Morgan-Kaufmann, pp.936-943, 1992.

[26] B. W. Mel,"The clusteron: toward a simple abstraction for a complex neuron," In *Advances in Neural Information Processing Systems 4*, S. J. Hanson, J. E. Moody and R. P. Lippmann (Eds.), pp.35-42, Morgan Kaufmann, 1992.

[27] B. W. Mel and C. Koch,"Sigma-pi learning: on radial basis functions and cortical associative learning," in *Advances in Neural Information Processing Systems 2* D.S. Touretzky (Ed.), Morgan-Kaufmann, pp.474-481, 1990.

[28] Y. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, 1989.

[29] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, Vol.3, pp.246-257, 1991.

[30] D. Psaltis, C. H. Park, and J. Hong, "Higher order associative memories and their optical implementation," *Neural Networks*, Vol.1, pp.149-163, 1988.

[31] H. L. Royden, *Real Analysis*, (2nd Ed.), MacMillan, 1968.

[32] T. D. Sanger, "A tree-structured adaptive network for function approximation in high-dimensional spaces," *IEEE Trans. on Neural Networks*, Vol.2, No.2, March 1991.

[33] T. D. Sanger, R. S. Sutton and C. J. Matheus, "Iterative construction of sparse polynomial approximations," in *Advances in Neural Information Processing Systems 4*, S. J. Hanson, J. E. Moody and R. P. Lippmann (Eds.), Morgan-Kaufmann, pp.1064-1071, 1992.

[34] Y. Shin and J. Ghosh, "The pi-sigma network: an efficient higher-order neural network for pattern classification and function approximation," *Proceedings of International Joint Conference on Neural Networks*, Vol.I, pp.13-18, Seattle, Washington, July 1991.

[35] Y. Shin and J. Ghosh, "Realization of boolean functions using binary pi-sigma networks," in *Intelligent Engineering Systems through Artificial Neural Networks*, C. H. Dagli, S. R. T. Kumara and Y. C. Shin (Eds.), ASME Press, pp.205-210, 1991.

[36] Y. Shin and J. Ghosh, "Approximation of multivariate functions using ridge polynomial networks," *Proceedings of International Joint Conference on Neural Networks*, Vol.II, pp.380-385, Baltimore, Maryland, June 1992.

[37] Y. Shin and J. Ghosh, "Function approximation using higher-order connectionist networks," Technical Report #TR-92-12-87, Computer and Vision Research Center, The University of Texas at Austin, May 1992.

[13] J. H. Friedman, W. Stuetzle and A. Schroeder, "Projection pursuit density estimation," *Journal of the Amer. Stat. Assoc.*, Vol. 79, No.387, pp.599-608, Sep. 1984.

[14] S. Geman, E. Bienenstock and R. Doursat, "Neural Networks and the bias/variance dilemma," *Neural Computation*, Vol.4, No.1, pp.1-58, 1992.

[15] J. Ghosh and A. C. Bovik, "Neural networks for textured image processing," in *Artificial Neural Networks and Statistical Pattern Recognition : Old and New Connections*, I. K. Sethi and A. K. Jain (Eds.), Elsevier Science Publishers, pp.133-154, 1991.

[16] J. Ghosh and Y. Shin, "Efficient higher-order neural networks for classification and function approximation," *International Journal of Neural Systems*, Vol.3, No.4, pp.323-350, 1992.

[17] C. L. Giles and T. Maxwell, "Learning, invariance, and generalization in a high-order neural network," *Applied Optics*, Vol.26, No.23, pp.4972-4978, 1987.

[18] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, Vol.2, pp.359-366, 1989.

[19] J.-N. Hwang, H. Li, M. Maechler, R. D. Martin and J. Schimert, "A comparison of projection pursuit and neural network regression modeling," in *Advances in Neural Information Processing Systems 4*, S. J. Hanson, J. E. Moody and R. P. Lippmann (Eds.), Morgan-Kaufmann, pp.1159-1166, 1992.

[20] P. J. Huber, "Projection pursuit," *The Annals of Statistics*, Vol.12, pp.435-475, 1985.

[21] A. G. Ivakhnenko,"Polynomial theory of complex systems,"*IEEE Trans. on Systems, Man, and Cybernetics*, Vol.1, No.4, pp.364-378, Oct. 1971.

[22] C. Koch and T. Poggio, "Multiplying with synapses and neurons," in *Single Neuron Computation*, T. McKenna, J. Davis and S. F. Zornetzer (Eds.), Academic Press, 1992.

[23] Y. LeCun, J. S. Denker and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (Ed.), Morgan-Kaufmann, pp.598-605, 1990.

[24] M. R. Lynch and P. J. Rayner, "The properties and implementation of the non-linear vector space connectionist model," *Proceedings of First IEE International Conference on Artificial Neural Networks*, London, England, pp.184-190, Oct. 1989.

[25] J. McClelland and D. Rumelhart, *Parallel Distributed Processing Vol.1*, The MIT Press, 1987.

References

[1] S. Beck, L. Deuser, R. Still and J. Whiteley. "A hybrid neural network classifier of short duration acoustic signals," *Proceedings of International Joint Conference on Neural Networks*, Vol.I, pp.119-124. Seattle, Washington, July 1991.

[2] H. H. Chen, Y. C. Lee, T. Maxwell, G. Z. Sun, H. Y. Lee and C. L. Giles, "Higher order correlation model for associative memory," *AIP Conference Proceedings 151*, pp.86-99, 1986.

[3] C. K. Chui and X. Li, "Realization of neural networks with one hidden layer," Technical Report #CAT-244, Center for Approximation Theory, Department of Mathematics, Texas A&M University, March 1991.

[4] J. M. Combes, A. Grossman and Ph. Tchamitchian (Eds.), *Wavelets: Time-Frequency Methods and Phase Space*, Springer-Verlag, 1989.

[5] N. E. Cotter, "The Stone-Weierstrass theorem and its application to neural networks,"*IEEE Trans. on Neural Networks*, Vol.1, No.4, pp.290-295, 1990.

[6] G. Cybenko, "Approximations by superposition of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, Vol.2, pp.303-314, 1989.

[7] P. J. Davis, *Interpolation and Approximation*, Dover, 1975.

[8] R. Durbin and D. E. Rumelhart, "Product units: a computationally powerful and biologically plausible extension to backpropagation networks," *Neural Computation*, Vol.1, pp.133-142, 1989.

[9] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (Ed.), Morgan-Kaufmann, pp.525-532, 1990.

[10] J. A. Feldman and D. H. Ballard, "Connectionist models and their properties," *Cognitive Science*, Vol.6, pp.205-254, 1982.

[11] J. H. Friedman, "Adaptive techniques for machine learning and function approximation," Invited Talk, NATO ASI Workshop, "From Statistics to Neural Networks", Les Arcs, France, June 1993.

[12] J. H. Friedman and W. Stuetzle, "Projection pursuit regression," *Journal of the Amer. Stat. Assoc.*, Vol.76, No.376, pp.817-823, Dec. 1981.

Acknowledgement

This work is supported in part by ONR contracts N00014-89-C-0298 and N00014-92C-0232, by AFOSR grant F49620-93-1-0307, and by a faculty assistantship from TRW. We wish to thank Sungbin Im for preparing surface plots used in this paper, Kagan Tumer for providing simulation results of the multilayered perceptron, optimal brain damage and cascade-correlation algorithm on the oceanic signals classification, and the anonymous reviewers for helpful comments.

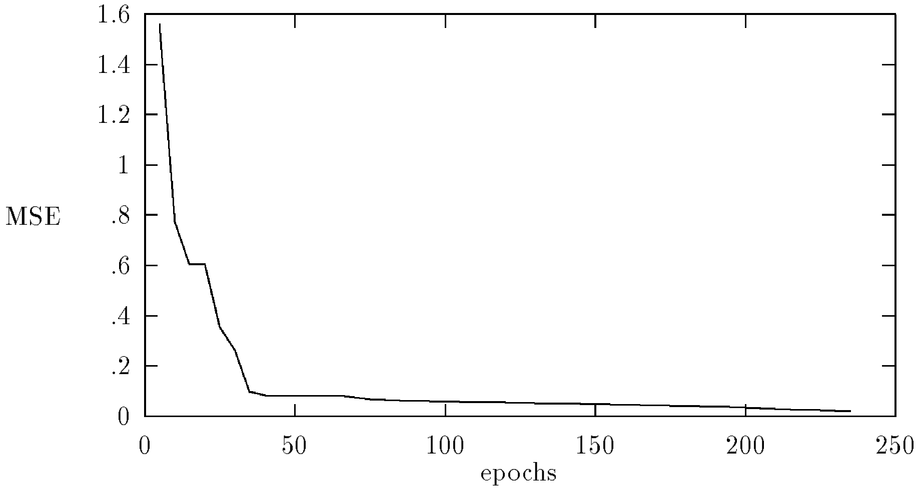


Figure 10: The learning curve for the realization of multivariate polynomial using the RPN.

5 Concluding Remarks

The ridge polynomial network (RPN) is based on a generalization of both Chui and Li's multivariate polynomial representation theorem [3] and the pi-sigma network [16, 34, 35]. We consider it as a special form of projection pursuit regression that is amenable to adaptive, incremental, supervised learning.

Simulation results on a surface fitting problem show that the constructive learning algorithm proposed for the RPN produces a smooth generalization which is a desirable property in multivariate function approximation. Successive incorporation of higher-order terms during learning captures the higher-order variations in the problem. A key feature of RPN network growth is that it takes place naturally as part of the training process, while maintaining a regular structure. It is widely acknowledged that there is no single method - statistical or neural - that gives the best results for *all* function approximation problems [11]. Thus, while our simulations demonstrate several desirable properties of RPNs, a deeper understanding of the classes of functions for which they are natural can only be obtained after extensive comparative experiments with several different techniques and data sets.

The difficult theoretical question of whether the constructive learning algorithm can achieve the same or similar approximation capability as an adequately large fixed structure is left as future work. Also, optimization techniques like "orthogonalization" of polynomial bases can be investigated to improve the performance of the networks without substantially increasing their complexity.

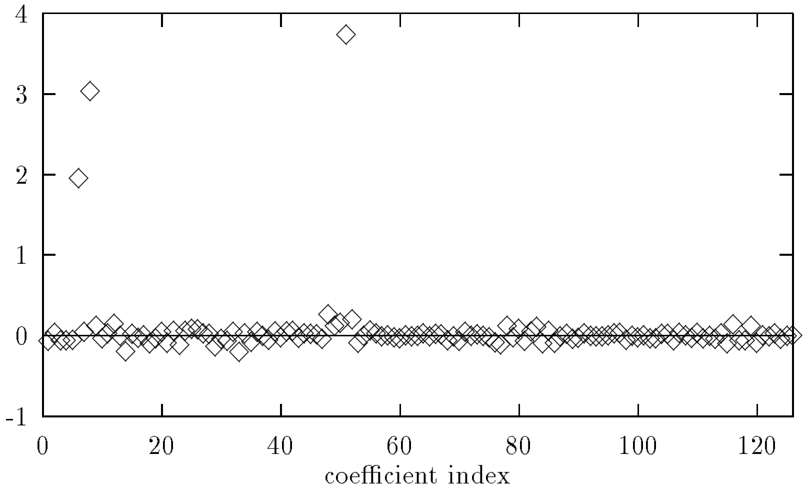


Figure 8: The coefficients obtained by the RPN of degree up to 4. The "coefficient index" just indicates each coefficient out of the total of 126 coefficients after expanding the RPN. The 6th, 8th and 51st indices correspond to constant, x_1x_2 and $x_3x_4x_5$ terms, respectively.

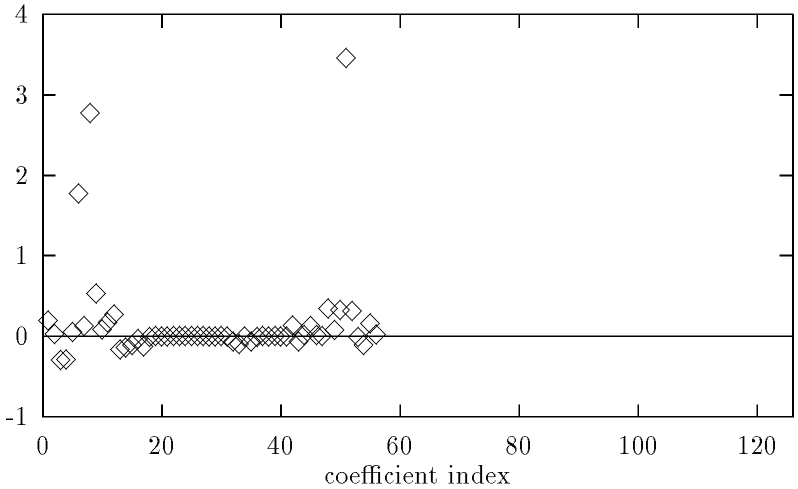


Figure 9: The coefficients obtained by the RPN of degree up to 3. There are the total of 56 coefficients after expanding the RPN. As in the case of 4th degree RPN, the 6th, 8th and 51st indices correspond to constant, x_1x_2 and $x_3x_4x_5$ terms, respectively.

4.3 Realization of Multivariate Polynomial

Finally, the RPN was trained to realize a multivariate polynomial $f : [-1, 1]^5 \rightarrow \Re$ which was given by[4]:

$$f(x_1, x_2, x_3, x_4, x_5) = 2 + 3x_1x_2 + 4x_3x_4x_5. \tag{21}$$

Unlike the other two classification/regression problems considered, this simulation mainly focuses on the *realization* of an unknown polynomial since the RPN uses special forms of polynomials. Note that we do not expect *exact* representation because of the limited number of training data and the nature of the least square learning used in the incremental learning algorithms.

We randomly selected 200 input points from $[-1, 1]^5$ and used them as a training set. The weights were initialized by uniformly-deviated random numbers in [-1,1]. The parameters η, r, δ_η and δ_r were set to 0.02, 0.0001, 0.8 and 0.1, respectively. Since the emphasis was on the possibility of the *representation*, we did not have a testing set to test generalization. The RPN was trained starting from the 1st degree initial PSN (i.e., linear perceptron) with linear output activation function.

The 2nd, 3rd and 4th degree PSNs were added at 6th, 22nd and 65th epochs, respectively, and the training finished at 238th epoch with MSE of 0.02. The final polynomial $\hat{f}$ realized by the RPN after training can be expanded as an ordinary multivariate polynomial as follows:

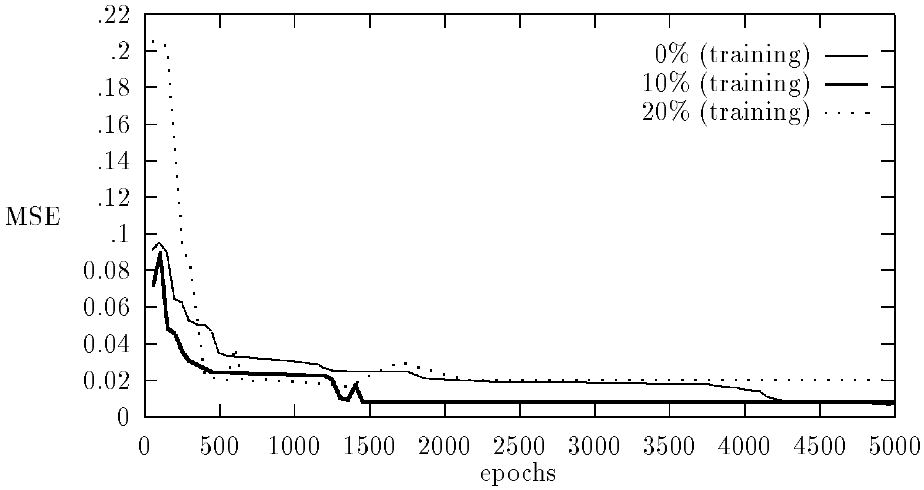
$$\hat{f}(x_1, x_2, x_3, x_4, x_5) \approx 1.96 + 3.03x_1x_2 + 3.74x_3x_4x_5. \tag{22}$$

Figure 8 shows all 126 coefficients which were obtained by the RPN after expansion. Note that 126 coefficients correspond to all possible coefficients for 4th degree (ordinary) multivariate polynomial with 5 inputs (see Eq. 2). The largest unnecessary coefficient was 0.27 for the $x_3^2x_4$ term. This is below 10% of the smallest coefficient used in f (constant term 2). Note that although the algorithm added 4th degree PSN even though the highest order of f is 3, the 4th order terms obtained by the algorithm have negligible magnitudes. Also the representation achieved is very close to the original function f as given in Eq. 22, which demonstrates the capability of the algorithm even with a small number of training points. To see how incremental learning achieves better approximation through training, Figure 9 shows coefficients of up to 3rd degree PSNs at the 64th epoch, i.e. just before the 4th degree PSN was added. There are the total of 56 coefficients after expansion. Using PSNs of degrees up to 3, the realized function $\hat{f}$ was:

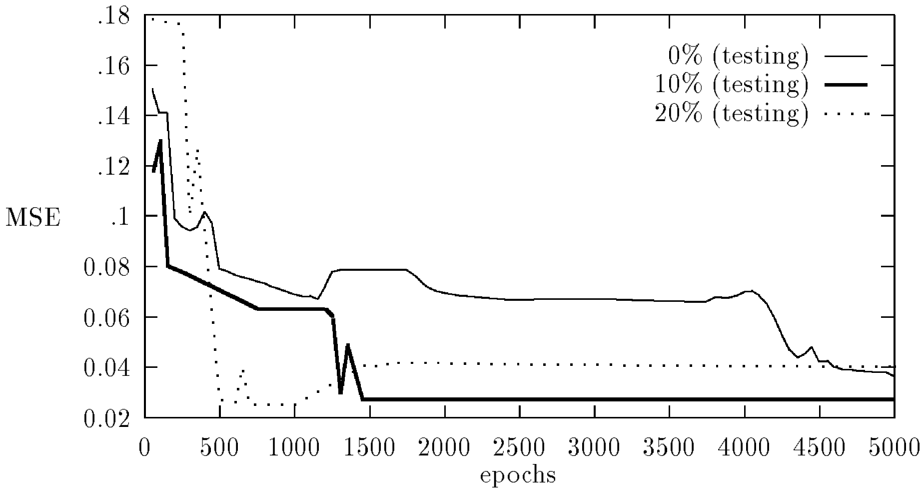
$$\hat{f}(x_1, x_2, x_3, x_4, x_5) \approx 1.76 + 2.77x_1x_2 + 0.53x_1x_3 + 3.45x_3x_4x_5. \tag{23}$$

As can be observed, addition of higher degree PSN rendered closer approximation to the function f to be realized. Figure 10 demonstrates stable learning during the training for the RPN.

[4]This polynomial was studied in [33].

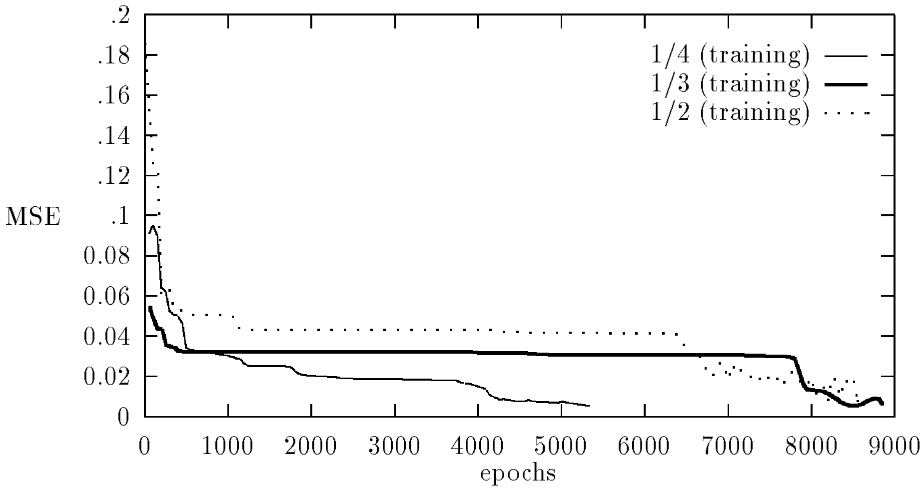


(a)

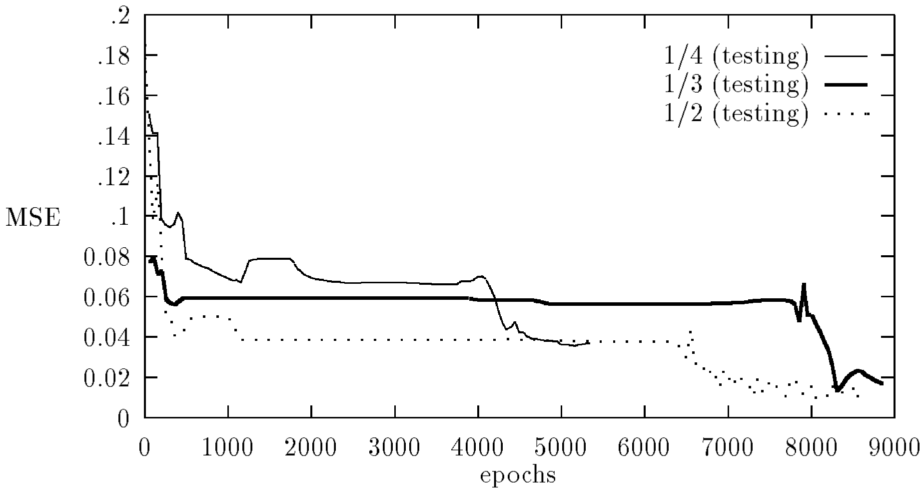


(b)

Figure 7: The learning curves for the approximation of Gabor function, using the training sets with different noise levels: (a) MSEs for the training sets, (b) MSEs for the testing sets.



(a)



(b)

Figure 6: The learning curves for the approximation of Gabor function, using the training sets of different sizes: (a) MSEs for the training sets, (b) MSEs for the testing sets.

	Size of the training set		
	1/4	1/3	1/2
η	0.4	0.3	0.12
r	0.005	0.005	0.005
δ_η	0.6	0.8	0.9
δ_r	0.1	0.1	0.1
Degree of initial PSN	4	4	4
Degree of final PSN	7	8	9
MSE(training)	0.005	0.006	0.008
MSE(testing)	0.037	0.016	0.008

Table 3: Simulation results using the RPN for the Gabor function approximation with different sizes of the training set.

	Noise (%)		
	0%	10%	20%
η	0.4	0.3	0.12
r	0.005	0.005	0.005
δ_η	0.6	0.8	0.98
δ_r	0.1	0.1	0.1
Degree of initial PSN	4	4	4
Degree of final PSN	7	10	6
MSE(training)	0.005	0.009	0.02
MSE(testing)	0.037	0.027	0.04

Table 4: Simulation results using the RPN for the Gabor function approximation with different noise levels on the training set.

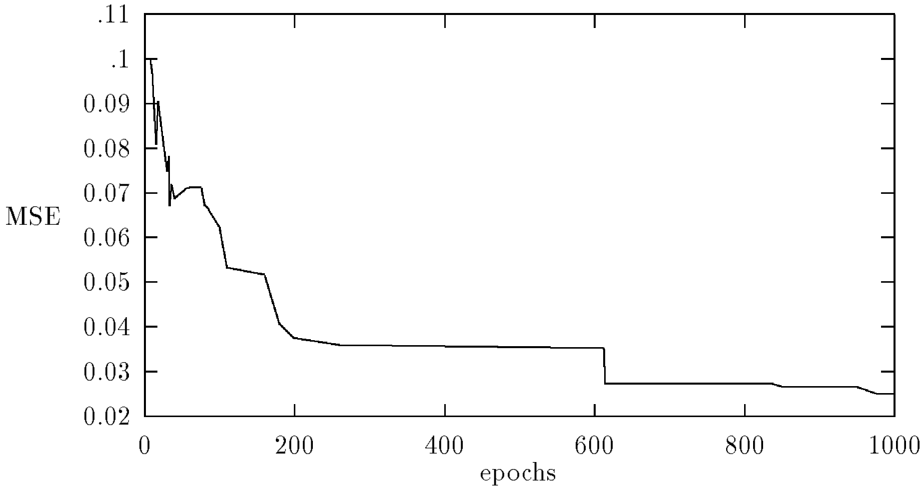


Figure 5: The learning curve for the approximation of Gabor function using the incremental learning algorithm for the RPN (up to 1000 epochs). Starting with a 4th degree PSN, the algorithm added the 5th and 6th degree PSNs at the 76th and 613th epoch, respectively.

Using smaller σ increases the dynamic range of the function. Out of 256 input points, we chose 1/4, 1/3 and 1/2 of 256 points as training set, corresponding to 64, 85 and 128 training points, respectively. As expected, the generalization capability indicated by the approximation accuracy increases for the testing set as more points are included in the training set, while it becomes worse for the training set. Table 3 summarizes the results. Figures 6 (a) and (b) show the learning curves for both training and testing sets for all three cases using the RPN.

To see how the algorithm performs for noisy data, we added noise to the training set. We chose 1/4 out of 256 input points as the training set as in the previous simulation. Then we added uniformly-deviated random noise to all these 64 training points. Testing data had no noise. The first case was adding noise from [-0.03,0.03] and the second case was adding noise from [-0.06,0.06]. The maximum values (0.03 and 0.06) are roughly corresponding to 10% and 20% of the average values of all training patterns, respectively. Table 4 shows the simulation results. We observe that with higher level of noise, the approximation accuracy for the training set becomes worse. However, slight addition of noise (10% case) increased the generalization capability. Figures 7 (a) and (b) show the learning curves for both training and testing sets for various noise levels using the RPN.

simulation, the Gabor function to be approximated was:

$$h(x_1, x_2) = \frac{1}{2\pi(0.5)^2} \cdot e^{-\frac{x_1^2+x_2^2}{2(0.5)^2}} \cdot \cos(2\pi(x_1 + x_2)). \tag{20}$$

Figure 4(a) shows the Gabor function used for the simulation as desired output. For training, 128 input points were randomly selected from 256 input points taken from an evenly spaced 16×16 grid on $[-0.5, 0.5]^2$. The remaining 128 points were used for testing. The RPN used a hyperbolic tangent activation function $\sigma(\cdot) = tanh(\cdot)$ as an output activation function for smoother generalization. All the initial weights used were randomly assigned to values between -0.5 and 0.5.

The parameters η, r, δ_η and δ_r were set to 0.4, 0.0001, 0.6 and 0.1, respectively. The RPN started the training with a 4th degree PSN, and the algorithm added the 5th and 6th degree PSNs at 76th and 613th epochs, respectively. The learning was quite stable and the MSE decreased when additional higher degree PSNs were co-opted, as shown in Figure 5. Figure 4(b) shows the actual network output of an RPN including up to 8th degree PSNs after 10000 epochs. MSEs obtained are 0.003 and 0.004 for the training and testing patterns, respectively. We observe that the RPN generalizes well and achieves a smooth interpolation among the training data points.

The Gabor function has been previously used in [16] to compare the Pi-Sigma network with an MLP. After experimenting with different numbers of hidden units and learning parameters, the best result for a two-layered MLP was obtained using 20 hidden units, with $\nu = 0.1$ and $\alpha = 0.9$. This configuration yielded an MSE of 0.012 for both training and test sets after 500 epochs. A 6th order HPU, in comparison, yielded MSE(training) of 0.028 and MSE(testing) of 0.037 after 5000 epochs.

(a) (b)

Figure 4: Approximation results: (a) Gabor function to be approximated, (b) output of the RPN.

To see the effect of the size of the training set, we have also tried various combinations of training/testing sets. We used a slightly different parameter value for σ, where the scale factor σ in the Gabor function was set to 0.42 rather than 0.5 as used in the previous simulation set (Eq. 20).

Networks	Epochs	Testing set correct classification rate	# Multiplications ($\times 10^6$)
RPN			
2nd order	50	84.1	31.9
2nd order	100	84.8	63.8
3rd order	200	83.6	191.7
3rd order	300	87.8	319.6
3rd order	390	88.6	434.7
Backpropagation			
10 hidden units	300	84.0	333.3
20 hidden units	300	89.2	666.6
40 hidden units	300	90.5	1333.1
80 hidden units	300	91.0	2666.2
OBD			
10 hidden units	300	86.2	401.2
20 hidden units	300	89.3	802.3
40 hidden units	300	90.1	1604.7
80 hidden units	300	89.7	3209.3
Cascade-correlation			
23.2 hidden units	300	88.3	221.2
27.4 hidden units	300	85.9	267.6

Table 2: Comparative results for classification of oceanic signals using the RPN, MLP with standard backpropagation algorithm, OBD and cascade-correlation algorithm. For OBD, "saliency" was calculated at every 10 epochs. The number of hidden units in the cascade-correlation algorithm is the average of 8 runs.

the first 20 epochs and 0.1 after that. The momentum factor was 0.9, and the initial weights were randomly selected from [-1,1]. For the training set, these networks showed quite reasonable results ranging from 90% to 100%.

We also compare the computational complexity of all the networks used in terms of the number of multiplications needed to finish the training, which has the same order as the number of additions. To obtain these numbers, we first note that each iteration, involving presentation of one training pattern, largely consists of two phases: (i) the forward phase involving calculation of network outputs and (ii) the backward phase for updating the weights. Thus, number of multiplications needed to finish the training can be obtained by multiplying number of multiplications per iteration by the number of training patterns and the number of epochs to finish the training. In this way, the comparison of each network's computational complexity can be done in a more equitable fashion, and with less dependence on the simulation environment. The calculation of error after each epoch, which is common to all three networks, is not considered. Also, we ignore the computation of the activation function, but note that this simplification favors the MLP type networks as they use a nonlinear activation function for the hidden units while the RPN does not.

As can be observed, the RPN achieved results comparable to other networks, but used much less computation and memory (number of units, weights). Note that we do not have to determine *a priori* the structure of the RPN as opposed to MLP and OBD in which we should first determine the number of hidden units. The cascade-correlation algorithm which is a constructive method avoids this problem, however, simulation results suggests that this algorithm sometimes overfits the problem (the classification accuracy of this algorithm for the training set was 100% using an average of 27.4 hidden units), and shows a wider variation in performance.

4.2 Multivariate Function Approximation

In this experiment, we aimed at observing how smooth RPN can generalize using a set of sparse data from an unknown multivariate function. A 2-D Gabor function was tried for this purpose.

It is well known that the highly-oriented simple cell receptive fields in the visual cortex of mammals can be closely modeled by 2-D Gabor functions, which are Gaussians modulated by sinusoidal functions. Gabor functions play an important role in physics, since they are the unique functions that achieve the lower bound for the space-frequency uncertainty product, which is a measure of a function's simultaneous localization in both spatial and frequency domains [15]. The convolution version of *complex* 2-D Gabor functions have the following form:

$$h(x_1, x_2) = \frac{1}{2\pi\lambda\sigma^2} \cdot e^{-\frac{(x_1/\lambda)^2 + x_2^2}{2\sigma^2}} \cdot e^{2\pi i (u_0 x_1 + v_0 x_2)}. \tag{19}$$

Here, λ is the aspect ratio, σ is the scale factor and (u_0, v_0) are modulation parameters. For

The following network parameters were used for the results reported in this paper: $\eta = 0.1, r = 0.0001, \delta_\eta = 0.8$ and $\delta_r = 0.1$. This set of parameters was chosen by several trials to yield the best performance on the testing sets. While this compromises the dataset's status as a true test set, it yields limits to how well the network can perform. For fair comparisons, parameters choices for other networks were also made in this fashion. The weights were initialized with uniformly-deviated random numbers in [-1,1]. The input vectors were of 25 dimensions and the output vectors were of 6 dimensions where each output represents one out of 6 signal types. We used the sigmoid for the output activation function, and let the desired output during training be 1 for the correct class and 0 otherwise. The training started with a 2nd degree PSN and added a 3rd degree PSN at the 200th epoch. Figure 3 shows classification accuracy for both training and testing sets when training epochs increase. The best result for the testing set was 88.6% (890 hits out of 1005) classification accuracy at the 390th epoch. For training set, the accuracy was 93.8% (647 hits out of 690). We could obtain better results for the training set using a longer training phase and addition of PSNs of degree higher than 3, however, the results for the testing set were worse than the one reported above, indicating over-training.

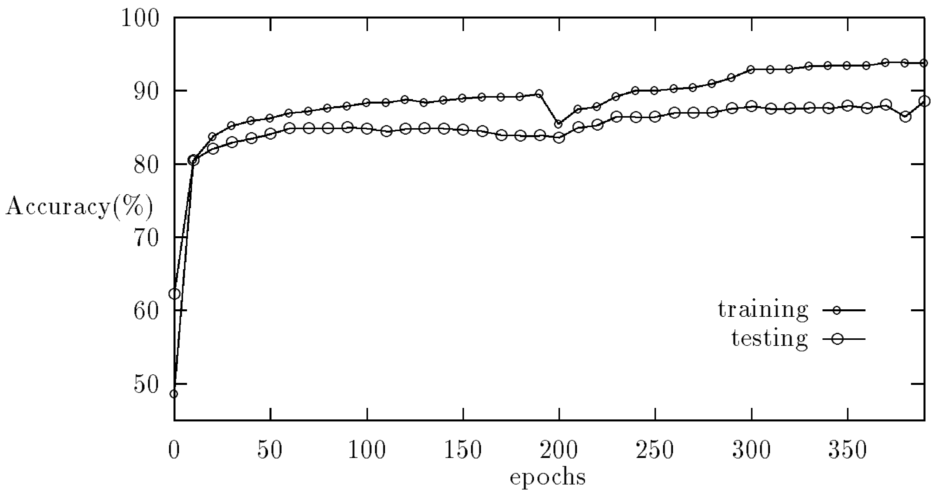


Figure 3: Classification accuracy for oceanic transient data using the RPN.

Table 2 compares the results using the RPN with those of finely-tuned MLP with standard backpropagation algorithm, optimal brain damage (OBD) [23] and cascade-correlation algorithm [9] reported in [41]. For MLP, OBD and cascade-correlation, the classification accuracies for the testing set were obtained after training up to 300 epochs. For these networks, the learning rate was 0.2 for

Class	Signal Description	Number of patterns	
		Training	Testing
A	Porpoise sound	116	284
B	Ice 1	116	175
C	Ice 2	78	39
D	Whale sound 1	116	129
E	Whale sound 2	148	251
F	Background noise	116	127
	Totals	**690**	**1005**

Table 1: Description of oceanic data set used in the simulation.

4 Simulation Results

To show the capability of the incremental learning algorithm for the RPN, we performed the following three experiments using the RPN with the constructive learning algorithm.

- Classification of transient oceanic signals,
- Approximation of two-dimensional (2-D) Gabor function (i.e., surface fitting), and
- Realization of a multivariate polynomial.

4.1 Classification of Transient Oceanic Signals

The data set used for the experiments consists of 25-dimensional feature vectors extracted from short-duration passive sonar signals due to six types of naturally occurring oceanic sources. The signal source type and number of training and testing patterns are given in Table 1.

Signal preprocessing as well as the extraction of good feature vectors is crucial to the performance of the classifiers[1]. The 25-dimensional feature vectors extracted from the raw signals consist of:

- 16 coefficients of Gabor wavelets - a multi-scale representation that does not assume signal stationarity[4],
- 1 value denoting signal duration, and
- 8 other temporal descriptors and spectral measurements.

The preprocessing techniques and choice of the feature vector reflect the following two goals; (i) all effects that vary, but not as a result of the events of interest, be removed or accounted for to the greatest possible extent, and (ii) the presence of each type of signal of interest should result in a measurable difference in the observable features.

where $\mathbf{z} = (1, \mathbf{x})$, $\mathbf{v}_j = (w_{j0}, \mathbf{w}_j) \in \Re^{d+1}$ are augmented input and weight vectors, respectively and $y_k^{'}$ is the first derivative of the network output.

4. At the end of every epoch (i.e., after applying the above procedure for all the training patterns), calculate the error for the current epoch, ϵ_c.

5. If $\epsilon_c < \epsilon_{th}$ or $t > t_{th}$, stop.

6. Else do:

   (a) If $|\frac{\epsilon_c - \epsilon_p}{\epsilon_p}| \geq r$ do:

   - $t \leftarrow t + 1$
   - $\epsilon_p \leftarrow \epsilon_c$
   - Go to Step 3.

   (b) Else do:

   - $\widehat{P_k} \leftarrow P_k$
   - $r \leftarrow r \cdot \delta_r$
   - $\eta \leftarrow \eta \cdot \delta_\eta$
   - $t \leftarrow t + 1$
   - $\epsilon_p \leftarrow \epsilon_c$
   - $k \leftarrow k + 1$
   - Go to Step 2.

An unknown function f is successively approximated by this incremental learning algorithm as:

$$f \approx \sigma(P_{k_0}),$$

$$f \approx \sigma(\widehat{P_{k_0}} + P_{k_0+1}),$$

$$f \approx \sigma(\widehat{P_{k_0}} + \widehat{P_{k_0+1}} + P_{k_0+2}),$$

$$\vdots$$

The number of parameters used in this algorithm is $\mathcal{O}(d \cdot k^2)$ if we start with $k_0 = 1$ and end up with the k-th degree PSN. Only the parameters of the most recently added PSN are adapted in a given epoch. This is contrary to the ordinary higher-order networks which adapt all parameters at each epoch.

After the set of weights $\mathbf{v}_j$ to be updated is chosen by any of above rules, an LMS-type approach is taken [44]. Here, the instantaneous squared error for the p-th pattern ϵ^p is used as an estimate of the true MSE. Applying gradient descent on the selected set of weights $\mathbf{v}_l$, one obtains:

$$\Delta w_{li} \propto -\frac{\partial \epsilon^p}{\partial w_{li}}, \quad i = 0, 1, \cdots, d, \tag{16}$$

leading to the following update rule:

$$\Delta \mathbf{v}_l = \eta \cdot (f^p - y^p) \cdot (y^p)' \cdot (\prod_{j \neq l} h_j^p) \cdot \mathbf{z}^p, \tag{17}$$

where $(y^p)'$ is the first derivative of the output, $\mathbf{z} = (1, \mathbf{x}) \in \Re^{d+1}$ is the augmented input pattern and η is the learning rate. Other sets of weights which are not selected remain the same. The methods are easily extensible for multiple outputs since each output has its own summing units.

3.2 Constructive Learning Algorithm for the RPN

As discussed earlier, the constructive learning algorithm for the RPN is based on the asynchronous update rule for the PSN. Before presenting the algorithm, we need the following notations — ϵ_{th}: threshold mean squared error (MSE) for the training phase, ϵ_c, ϵ_p: MSEs for the current epoch and previous epoch during the training, respectively, r: threshold for successive addition of a new PSN, η: initial learning rate, δ_r, δ_η: decreasing factors for r and η, respectively[3], k: degree of the PSN, P_k at the k-th algorithmic step, t: number of training epochs, t_{th}: threshold epoch to finish the training, $\widehat{P_k}$: P_k after its weights are frozen, and y_k: network output at the k-th algorithmic step, $= \sigma(\sum_{i=k_0-1}^{k-1} \widehat{P_i} + P_k)$.

Constructive Learning Algorithm for RPN

1. Initialization Step: Set $k = k_0$ (≥ 1), $\widehat{P_{k_0-1}} = 0$ and $t = 0$. Assign suitable values to $\epsilon_{th}, \epsilon_p, r, \eta, \delta_r, \delta_\eta$ and t_{th}.

2. Set $P_k(\mathbf{x}) = \prod_{j=1}^{k}(< \mathbf{w}_j, \mathbf{x} > + w_{j0})$.

3. For all training patterns do:

   (a) Calculate actual network output, y_k

   (b) Update weights by applying the asynchronous update rule as:

$$\Delta \mathbf{v}_l = \eta \cdot (f - y_k) \cdot y_k' \cdot (\prod_{j \neq l}^{k} (< \mathbf{w}_j, \mathbf{x} > + w_{j0}) \cdot \mathbf{z} \quad (l = 1, \cdots, k), \tag{18}$$

[3]These parameters are needed because the dynamic range of error becomes smaller with successive addition of higher-degree PSNs.

the RPN.

Learning algorithms for PSNs are based on gradient descent on the estimated mean squared error surface in weight space. For a PSN, the mean squared error (MSE) objective is as follows:

$$\epsilon = \sum_{p=1}^{L} \epsilon^p = \frac{1}{L} \sum_{p=1}^{L} (f^p - y^p)^2, \tag{15}$$

where f is the unknown desired function to be approximated, superscript p denotes the p-th training pattern, f^p is the desired output for the p-th pattern, $y^p = \sigma(\prod_j h_j^p)$, and the summation is over all L training patterns[2].

In typical feedforward networks, all the weights are modified at each update step. However, this updating scheme can lead to instability problems for the PSN unless the learning rate is sufficiently small. This is due to higher order cross-terms generated at the output product unit that may not be in the direction of steepest gradient. In [16], we formally show why the magnitude of the learning rate is limited by a *fully synchronous* updating scheme. Instead of modifying all the weight sets synchronously at each update step, one subset of weights chosen at a time for updating. We proposed the randomized [34] and asynchronous updating rules [35] as methods to choose weight sets to be updated.

- **Randomized rule:** Consider a PSN with d inputs and k summing units. At each update step, we randomly select a summing unit and only update the set of $d+1$ weights associated with its inputs.

- **Asynchronous rule:** At each update step, all k sets of weights are updated but in an asynchronous manner. That is, one set of weights

  $$\mathbf{v}_j = (w_{j0}, \mathbf{w}_j) = (w_{j0}, w_{j1}, \cdots, w_{jd})$$

  (corresponding to the j-th summing unit) is chosen at a time and modified according to the weight update rule. Then, for the same input pattern, the output is re-computed for the modified network, and the error is used to update a different set of weights. For every input, this procedure is performed k times so that all k sets of weights are updated once.

The improved properties of both of these approaches is explained by the analysis given in [16]. In this paper, we concentrate on the **asynchronous rule** which performs better than the randomized one in terms of training speed.

[2] For simplicity, a one dimensional output is considered. If multiple outputs are needed, the summation is also performed over all outputs and the final value is also divided by the number of output units to get the error measure. The extension is fairly straightforward, so we will concentrate only on the single output case.

interpretation of a combination of product and summing units in terms of a single neuron or a group of neurons. In particular, local regions of dendritic arbor could act as product units whose outputs are summed at the soma. Mel and Koch have also argued that sigma-pi units underlie the learning of nonlinear associative maps in cerebral cortex [27], and that aggregates (on the order of 100 cells) of pyramidal cells whose apical dendrites rise toward the cortical surface in tight clumps, act as a single virtual sigma-pi unit. A recent review of a number of different biophysical mechanisms that give rise to multiplicative interactions, can be found in [22]. These mechanisms range in specificity of interactions from pairs of individual synapses to small sets of neurons.

Mel has recently proposed the clusteron as an abstraction for a complex neuron that can extract higher order statistics from input stimuli [26]. In his model, a dendritic tree receives weighted synaptic contacts from a set of afferent axons. Each synaptic contact is given by a product of direct stimulus intensity and a weighted sum of neighborhood activity. We note that this description translates to an RPN, which can be considered as a mathematical abstraction of the clusteron model. However, certainly a biologically plausible network should have some self-organizing capabilities. This motivates investigation of Hebb-type learning schemes for the RPN in order to further pursue the plausibility issue.

3 Learning Algorithm for the RPN

Since each P_i in Eq. 11 is obtainable as the output of a PSN of degree i with linear output units, the learning algorithms developed for the PSN can also be used for the RPNs. So an immediate approach is to select the constituent PSNs and then adapt weights using gradient descent. However, the optimal network complexity, as determined by k, is often not known *a priori*. This has prompted several constructive (or destructive) algorithms that dynamically adapt network structure by adding (or pruning) weights or nodes until the most appropriate network size is obtained [41]. The trade-offs involved in network size vs. generalization capability can be understood in terms of the bias/variance dilemma [14]. As a network grows, model bias is decreased resulting in better generalization. However, if the number of free parameters is too large, model variance dominates and starts to degrade generalization performance. These considerations motivate us to develop a **constructive learning algorithm for the RPNs** in this section. This learning algorithm aims at approximating an unknown scalar function f in $\Re^d$ by successive addition of a PSN, P_i of higher degree than previously incorporated ones until a desired level of accuracy is attained.

3.1 Adaptation of Weights in Pi-sigma Networks

Before presenting the constructive learning algorithm for the RPN, we summarize the learning algorithms for the PSN which are a basic building block of the constructive learning algorithm for

The RPN is similar to GMDH and SONN in that the network is based on higher-order polynomials and is grown incrementally. Both GMDH and SONN lead to networks with arbitrary number of hidden layers and nodes in contrast to RPNs which have a well regulated architecture. Besides this difference and the difference in representation forms, the main trade-off is that RPN does not require extensive preprocessing using training data to come up with the desired structure, but the generality of the RPN structure also indicates that it may not be optimized for specific problems. Experimental results show that RPNs perform well over a broad class of problems, but may not be the best for a specific problem.

The LMS tree algorithm proposed by Sanger [32] also falls in this category if used with polynomial bases. The LMS tree starts with basis functions for an input component and grows a "subnetwork" under the basis function which has the maximum error variance. The subnetwork uses basis functions for a different input component, and thus successively forms basis functions which are factorized with respect to each input coordinate. Sanger *et al.* also proposed an iterative construction algorithm of sparse polynomial approximation which can be used in multi-dimensional input space [33]. The algorithm constructs a polynomial approximation of a given function from lower to higher order by "looking-ahead" relevant terms which will maximally contribute to decreasing the error if added. This algorithm is based on the LMS tree and a new figure-of-merit called "potential" is used for the look-ahead mechanism. These algorithms are different from the incremental algorithm of the RPN in the sense that they use different polynomial structures (factorized polynomials in the LMS tree and ridge polynomials in the RPN) and network growing algorithms. They also need extra processing for network growing mechanism. As in the case of GMDH and SONN, these algorithms yield irregular and problem-specific network structure as compared to RPN's highly regular structure.

2.4.3 Higher-order correlations in recurrent nets

The use of polynomial threshold elements of degree d ($d \geq 2$) has been considered in the context of Hopfield-type networks [42]. An extension of the outer-product storage rule to incorporate higher order correlations has also been proposed [2]. These studies pertain to fully connected recurrent networks and are mainly concerned with increasing storage capacity when such nets are used as associative memories. Thus they are fundamentally different from RPNs in structure and purpose, and are not considered any further.

2.5 Neurobiological Plausibility of RPNs

Are RPN type computing structures even remotely neurobiologically plausible? In neurophysiology, the possibility that dendritic computations could include local multiplicative nonlinearities is widely accepted [27]. Indeed, Durbin and Rumelhart [8] observe that there is a natural neurobiological

suitable thresholds for adding another PSN, which is much simpler. Thus even the order may not be selected *a priori*, and rather be determined by the data.

In system identification theory, Eq. 1 is identical to a Gabor-Kolmogorov polynomial expansion for a system [24], provided a linear activation function $\sigma(\cdot)$ is used. Here, the goal is to achieve polynomial approximation of the function using this model. A study of polynomial discriminant functions based on Eq. 1 with a thresholding transfer function, can also be found in classical pattern recognition literature [39]. Such functions have been revisited by Psaltis *et al.* [30] who study the increase in memory capacity because of the presence of more degrees of freedom, and propose optical implementations. Since RPNs are founded on Eq. 8, they are closer to projection pursuit regression, as discussed in Sec. 2.2, and have properties that are quite different from approaches based on Eq. 1 when the order of approximation is finite.

2.4.2 Incremental polynomial approximation

An alternative to parametric approaches is to use incremental techniques in which the number of free parameters is gradually increased, if need be. Ivakhnenko has proposed a polynomial approximation algorithm, called the group method of data handling (GMDH) algorithm, for complex nonlinear systems [21]. This algorithm is basically an incremental algorithm for building an approximation of the unknown system by beginning with simple polynomials and proceeding to more and more complex polynomial expressions. Its functioning can be expressed in terms of the incremental construction of a feedforward network in which the output of each node is a product of two variables, and whose parameters are obtained by regression. Starting with the input nodes, nodes are created in the next layer by pairwise connecting the inputs of the previous layer. The power of the method arises from the use of simple low-dimensional functions, and the ability of the algorithm to discard nodes with little promise. A heuristic that evaluates how close the realized function describes the training data in a least-mean-squares sense, is used to terminate and evaluate the algorithm. Unfortunately, this approach often leads to suboptimal structures because of its heuristic nature.

Another similar approach is the self-organizing neural network (SONN) algorithm [40]. This algorithm constructs a network, chooses the activation functions and adjusts the weights to incrementally build a model of the unknown system using representations selected from a pre-determined set of polynomials. The algorithm shares the spirit of GMDH type algorithms, but the use of a modified minimum description length criterion in conjunction with stochastic search based on simulated annealing for selection of node transfer functions leads to models that are simpler and more accurate. Comparisons with MLP given in [40] for the Mackey-Glass chaotic time series prediction problem show that the number of epochs required is less by an order of magnitude. This advantage is more than offset by the long times taken per epoch, largely due to simulated annealing search, which also degrades rapidly in time and quality with increasing problem size.

To counter the drawbacks of sigma-pi units, Durbin and Rumelhart [8] proposed the use of product units, each of which could realize a *single* higher order term, i.e., the output $z_j : \Re^d \rightarrow \Re$ of the j-th product unit is given by:

$$z_j(x_1, \cdots, x_d) = \prod_{i=1}^{d} x_i^{p_{ji}}. \tag{13}$$

The novelty lay in the fact that the exponents p_{ji}s could also be adapted using gradient descent, and were not limited to integer values. The authors envisoned that the product units would be mixed with summing units to form weighted sums of arbitrary products, and focussed on networks with a single hidden layer of product units, with output $y : \Re^d \rightarrow \Re$ given by:

$$y(x_1, \cdots, x_d) = \sum_j w_j z_j(x_1, \cdots, x_d) \tag{14}$$

with $w_j \in \Re$. The resulting networks are efficient for problems where a suitable discriminating function can be easily expressed in a (real-valued) sum of products form. For example, if each input component is ± 1, parity is simply obtained by checking whether the product of all the input components is $+1$ or not. As expected, product units are efficient for parity and symmetry problems.

The application of product unit networks to continuous valued problems has not been investigated much, either in [8] or in later works. We note that their success depends on whether the number of product units chosen is appropriate or not. This number may have little to do with the order of the function. Also, a network based on Eq. 14 is slowed because of backpropagation of error. In contrast, in the constructive learning scheme for the RPN which will be described in section 3, PSNs of increasing degrees are sequentially added during training until a desired level of performance is obtained. Thus little *a priori* knowledge of the desired function is required. Also, the RPN has only a single layer of adaptive weights as in the HPU.

The functional link architecture has also been proposed for generating higher-order or non-linear functions of the input components [28]. A functional link architecture estimates an unknown function as $\sum_i w_i \phi_i(\cdot)$, where $\phi_i(\cdot)$s are input components, products of input components, or other suitable functions such as sinusoidal functions. This approach is essentially linear regression, with $\phi_i(\cdot)$s serving as the basis functions, and has been well studied in several disciplines. In [28], excellent results are obtained for the 3-parity problem by choosing $x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3$, and $x_1x_2x_3$ as the basis functions. Clearly, *a priori* knowledge has been used to choose such an optimal basis. In another example on approximating a "sinusoidal" surface, the basis used was $x, \sin(\pi x), \cos(\pi x), \cdots$ and not a single higher-order polynomial was used. Surely this basis would not work well for the parity problem. Thus the functional link network does not answer the key issue of how to determine a suitable basis for arbitrary functions. As mentioned earlier, the issue when using RPN is to select

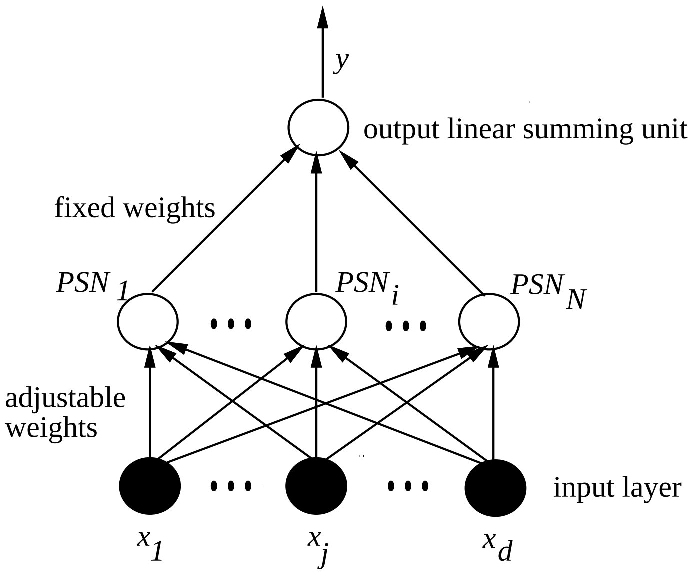


Figure 2: A ridge polynomial network. Each PSN_i represents an i-th degree pi-sigma network with linear output activation function.

where

$$P_i(\mathbf{x}) = \prod_{j=1}^{i} (< \mathbf{w}_j, \mathbf{x} > + w_{j0}), \ \ i = 1, \cdots, k. \tag{12}$$

In Eq. 12, $\sigma(\cdot)$ is a suitable (linear or nonlinear) activation function and $\mathbf{w}_j \in \Re^d$ and $w_{j0} \in \Re$ are determined by any suitable learning process. A constructive learning algorithm based on gradient descent is given in Section 3.

2.4 Relation to Other Polynomial Approximation Techniques

Several types of higher-order networks and polynomial-based approximation techniques can be found in neural network, system theory and estimation theory literature. In this subsection, we summarize notable alternatives and compare them with RPNs to highlight key differences.

2.4.1 Feedforward networks with product units and static architectures

Besides the HPU networks, this category includes several models in artificial neural network and system theory that are closely related. One such model is the "sigma-pi unit" [25], first used in a connectionist framework by Feldman and Ballard [10]. Since in this model, sums of products of *selected* input components are utilized, these units are a special case of HPUs. However, since this model does not allow terms like x_i^k $(k > 1)$, a single layer of sigma-pi units is not capable of approximating some functions well [5].

for some continuous functions g_i in one variable, and $a_i \in \Re, \mathbf{w}_i \in \Re^d$. The determination of g_i is usually done by one dimensional data smoothing over a certain bandwidth of projection, and a_i and $\mathbf{w}_i$ is usually determined by optimizing certain performance indices. Projection pursuit is a good example where parameterized $(a_i, \mathbf{w}_i)$ and nonparameterized (g_i) methods are combined. Researchers have shown that projection pursuit networks based on Eq. 10 work better on angular smooth functions than on Laplacian smooth functions [45]. Also, a learning rule taken from projection pursuit regression literature has been compared with backpropagation for training MLP networks [19]. In these studies, the projection pursuit learning achieves more parsimonious approximation (i.e., fewer number of hidden units for a given approximation accuracy) for many kind of nonlinear functions as compared to the standard backpropagation learning for the MLP.

The RPN and projection pursuit are similar in spirit. In fact, Eq. 8 is considered as a special case of projection pursuit regression with fixed functions g_i. Since projection pursuit finds the kernels g_i through more data pertinent smoothing methods, it realizes more parsimonious structure as compared to the RPN which uses fixed polynomial functions. However, the kernels g_i need to be obtained by nonparameterized methods in projection pursuit, which is not the case in RPN.

2.3 Network Architecture

Neural network architectures are either *fixed* or *dynamic* in size. In a fixed-size architecture, the size of an instance of a network in terms of number of units and weights, does not change from its first construction. On the other hand, in a dynamic-size architecture, the size of a network can be changed during the learning process. In particular, a *constructive learning (incremental learning)* method grows the network structure from a small, basic network for a given problem and the network becomes larger as learning proceeds until the desired level of approximation error is attained. The "cascade-correlation learning" [9] is an example of the constructive learning method. The main advantage of a constructive learning algorithm is that the networks realize a parsimonious approximation of the problem, typically resulting in better generalization.

Figure 2 shows a generic network architecture of the RPN using PSNs as basic building blocks. Note that the RPN has only a single layer of adaptive weights which has benefits in terms of training speed.

Instead of a linear activation function, we can use a nonlinear function such as sigmoid or hyperbolic tangent, in practical applications. Thus, an unknown scalar function f in $\Re^d$ can be approximated by the direct use of the RPN of degree up to k based on:

$$f(\mathbf{x}) \approx \sigma(\sum_{i=1}^{k} P_i(\mathbf{x})), \tag{11}$$

The uniform approximation capability of the ridge polynomials which have the form of Eq. 8 is immediate:

Proposition 1 *Let $f \in C(K)$ and $\epsilon > 0$ be given. Then there exists a ridge polynomial $p \in C(K)$ of the form Eq. 8 such that $|f(\mathbf{x}) - p(\mathbf{x})| < \epsilon$ for all $\mathbf{x} \in K$.*

Proof: By the generalized Weierstrass polynomial approximation theorem [7] (see Appendix B), there is a polynomial $p \in C(K)$ of the form Eq. 6 such that $|f(\mathbf{x}) - p(\mathbf{x})| < \epsilon$ for all $\mathbf{x} \in K$. Thus, the result follows from Theorem 2. □

2.2 Definition of Ridge Polynomial Networks

A *ridge polynomial network (RPN)* is defined as a feedforward network based on Eq. 8. An unknown function f defined on a compact set $K \subset \Re^d$ can be approximated by the RPN as:

$$\begin{aligned} f(\mathbf{x}) \approx \; & (<\mathbf{x}, \mathbf{w}_{11}> + w_{11}) + (<\mathbf{x}, \mathbf{w}_{21}> + w_{21}) \cdot (<\mathbf{x}, \mathbf{w}_{22}> + w_{22}) + \cdots \\ & + (<\mathbf{x}, \mathbf{w}_{N1}> + w_{N1}) \cdots (<\mathbf{x}, \mathbf{w}_{NN}> + w_{NN}). \end{aligned} \tag{9}$$

Note that each product term in Eq. 9 is obtainable as the output of a pi-sigma network (PSN) with linear output units. Thus, this new formulation can be considered to be a generalization of the PSNs. The total number of coefficients or weights in Eq. 7 and Eq. 8 is $\mathcal{O}(d \cdot N)$ and $\mathcal{O}(d \cdot N^2)$, respectively. In terms of total number of weights, Eq. 8 is less efficient than Eq. 7. However, the more regular and systematic structure of Eq. 8 over Eq. 7 makes it a more suitable basis for a feedforward network. Since, in general, there may not be much *a priori* information about the function to be approximated, it is difficult to choose an appropriate network size based on Eq. 7. On the other hand, an RPN based on Eq. 8 can be incrementally grown to meet a pre-defined error criterion. This idea is discussed again in the following section.

The RPN is closely related to a statistical regression technique called projection pursuit regression [20]. Loosely speaking, projection pursuit aims to systematically find "good" low dimensional projections of a high dimensional data points by optimizing a certain performance objective function. Projection pursuit is a powerful method which can avoid the curse of dimensionality [20] by working on low dimensional projections. A projection pursuit technique typically uses ridge functions to approximate a multivariate function f by forming a summation (projection pursuit regression [12]) or product (projection pursuit density estimation [13]). The basic formulation of projection pursuit regression is as follows:

$$f(\mathbf{x}) \approx \sum_{i=1}^{k} a_i g_i(<\mathbf{x}, \mathbf{w}_i>), \tag{10}$$

for some $a_{ij} \in \Re$ and $\mathbf{w}_{ij} \in \Re^d$. The following theorem by Chui and Li [3] states that any polynomial in $\Re^d$ can be expressed in terms of a ridge polynomial.

Theorem 1 (Chui & Li [3]) *Let Π_k^d denote the set of all polynomials in $\Re^d$ with degree $\leq k$, and let $\mathbf{x}^{\mathbf{i}_{jm}}$ represent a polynomial in d variables of total degree j, i.e., $\mathbf{x}^{\mathbf{i}_{jm}} = x_1^{i_{jm1}} \cdots x_d^{i_{jmd}}$, with $= i_{jm1} + \cdots + i_{jmd} = j$ for any m.*

For any polynomial

$$p(\mathbf{x}) = \sum_{j=0}^{k} \sum_{m=1}^{n_j} c_{jm} \mathbf{x}^{\mathbf{i}_{jm}}, \tag{6}$$

in Π_k^d where $c_{jm} \in \Re$, there exist constants $a_{jm} \in \Re$ and $\mathbf{v}_{jm} \in \Re^d$ such that

$$p(\mathbf{x}) = \sum_{j=0}^{k} \sum_{m=1}^{n_j} a_{jm} < \mathbf{x}, \mathbf{v}_{jm} >^j, \tag{7}$$

where $n_j = \begin{pmatrix} d+j-1 \\ j \end{pmatrix}$, is the total number of j-th degree polynomials in d variables. That is, $\sum_{m=1}^{n_j} c_{jm} \mathbf{x}^{\mathbf{i}_{jm}}$ in Eq. 6 denotes all j-th order terms of d input variables.

Basically, Eq. 6 means that $p(\cdot)$ is a polynomial of d variables comprising of all the terms from a constant to k-th order terms of d variables. The original form of the theorem is more complicated than the one presented here. In the original theorem, constants a_{jm} are uniquely determined from the linear equations. However, since we are only concerned with the *existence of a representation* of multivariate polynomials in terms of ridge polynomials, a simpler statement is adopted.

Based on Theorem 1, we formulate a more general and efficient ridge polynomial. The result is presented in the following theorem and a detailed proof can be found in Appendix A. The theorem implies that a new polynomial structure (i.e., ridge polynomial network) can be formed by addition of PSNs of different degrees, and this new structure has the same approximation capability as that of ordinary multivariate polynomials. The proof is obtained by showing that the ridge polynomial given by the right-hand-side of Eq. 7 is a special case of the RPN.

Theorem 2 *For any polynomial*

$$p(\mathbf{x}) = \sum_{j=0}^{k} \sum_{m=1}^{n_j} c_{jm} \mathbf{x}^{\mathbf{i}_{jm}},$$

in Π_k^d where $c_{jm} \in \Re$, there exist $w_{ji} \in \Re$ and $\mathbf{w}_{ji} \in \Re^d$ such that

$$p(\mathbf{x}) = \sum_{j=1}^{N} \prod_{i=1}^{j} (< \mathbf{x}, \mathbf{w}_{ji} > + w_{ji}), \tag{8}$$

where $N = \sum_{l=0}^{k} n_l$.

In this paper, we present a polynomial connectionist network called *ridge polynomial network* (RPN) that has good mapping capabilities in the sense that any continuous function on a compact set in $\Re^d$ can be uniformly approximated by this network. The RPN is efficient in the sense that they utilize univariate polynomials which are easy to handle. This fact is contrary to the higher-order feedforward networks in which direct use of multivariate polynomials causes an explosion of free parameters. Also, this network leads to highly regular structure as compared to the ordinary higher-order networks while maintaining their fast learning property.

The RPN is a generalization of the PSN that uses a special form of ridge polynomials. A representation theorem is introduced to show that any multivariate polynomial can be represented in terms of this ridge polynomial, and realized by an RPN. Approximation capability of the RPNs is shown by this representation theorem and the Weierstrass polynomial approximation theorem. The RPN provides a natural mechanism for incremental network growth, and we develop a constructive learning algorithm for the network.

In the following section, the RPN is formulated [36, 37]. Its approximation capability is shown by using Chui and Li's representation theorem [3] and the Weierstrass polynomial approximation theorem. Also, we compare RPN with other polynomial approximation techniques, and discuss neurobiological plausibility of the RPN. In section 3, we develop a constructive learning algorithm for the RPN by which a parsimonious approximation of an unknown function in terms of network complexity can be achieved. Section 4 presents simulation results on oceanic transient signal classification, a surface fitting problem and the realization of a multivariate polynomial using the constructive learning algorithm for the RPN. Section 5 concludes the paper.

2 Ridge Polynomial Networks

2.1 Ridge Polynomials

Let $C(K)$ be the vector space of continuous scalar functions on a compact subset K of a euclidean space $\Re^d$ $(d \geq 1)$ such as a hypercube $[0,1]^d$. For $\mathbf{x} = (x_1, \cdots, x_d)$ and $\mathbf{w} = (w_1, \cdots, w_d) \in \Re^d$, we denote $< \mathbf{x}, \mathbf{w} >$ as their inner product

$$< \mathbf{x}, \mathbf{w} > = \sum_{i=1}^{d} x_i w_i .$$

For a given compact set $K \subset \Re^d$, all functions defined on K in the form of $f(< \cdot, \mathbf{w} >) : K \rightarrow \Re$, where $\mathbf{w} \in \Re^d$ and $f(\cdot) : \Re \rightarrow \Re$ is continuous, are called ridge functions. A *ridge polynomial* is a ridge function that can be represented as:

$$\sum_{i=0}^{n} \sum_{j=1}^{m} a_{ij} < \mathbf{x}, \mathbf{w}_{ij} >^i , \tag{5}$$

To circumvent this problem, Shin and Ghosh [16, 34, 35] proposed a new class of polynomial feedforward neural networks called *pi-sigma networks (PSNs)* which use *products of sums* of input components instead of sums of products as in HPUs or "sigma-pi" units. The primary motivation was to develop a systematic method for maintaining the fast learning property and powerful mapping capability of single layer higher-order networks while avoiding the combinatorial increase in the number of weights required. A PSN is a feedforward network with a single "hidden" layer and product units in the output layer. Figure 1 shows a PSN with single output.

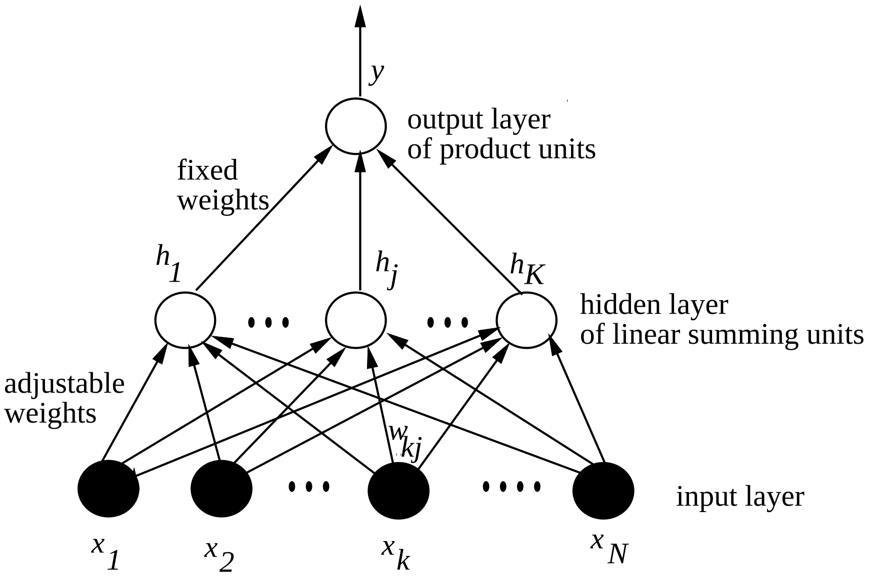


Figure 1: A pi-sigma network with one output.

PSNs have a highly regular structure and need a much smaller number of weights compared to other single layer higher-order networks. The presence of only one layer of adaptive weights results in fast training. A given scalar function f in $\Re^d$ is approximated by a k-th degree PSN as:

$$f(x_1, \cdots, x_d) \approx \sigma(\prod_{i=1}^{k} \sum_{j=1}^{d} (w_{ij} x_j + w_{i0})), \tag{3}$$

where $w_{ij} \in \Re$ are adjustable weights. The total number of weights for a k-th degree PSN in d-dimensional input space is only

$$(d+1) \cdot k. \tag{4}$$

The price paid for using a fewer number of weights is that the PSN is not a universal approximator. In fact, a k-th degree PSN realizes a *constraint* representation of a k-th degree HPU. However, empirical studies show that it is a good model for *smooth* functions and gives good results on various applications [16, 34, 38].

1 Introduction

Several feedforward neural networks provide mapping capabilities for approximating reasonable functions. These include the multilayered perceptron (MLP) which can approximate reasonable functions to any desired degree of accuracy using only one hidden layer of units or nodes having sigmoid activation functions, provided that sufficiently many hidden nodes are available [6, 18]. Similarly, Park and Sandberg [29] recently showed that the radial basis function networks having one hidden layer are capable of universal approximation in the sense that they can approximate reasonable functions very well in terms of various metrics.

Another class of feedforward networks are the polynomial neural networks which use higher-order correlations of the input components to perform nonlinear mappings using only a single layer of units. Examples of polynomial neural networks include higher-order networks [17], sigma-pi networks [25], product units [8] and some functional link architectures [28]. The basic building block of higher-order networks is the k-th degree *higher-order processing unit (HPU)* with which a given function $f : \Re^d \rightarrow \Re$ is approximated as [17, 25]:

$$\begin{aligned} f(x_1, \cdots, x_d) \quad \approx \quad & \sigma(w_0 + \sum_{i_1=1}^{d} w_{i_1} x_{i_1} + \sum_{i_1=1}^{d} \sum_{i_2=i_1}^{d} w_{i_1 i_2} x_{i_1} x_{i_2} + \cdots \\ & + \sum_{i_1=1}^{d} \sum_{i_2=i_1}^{d} \cdots \sum_{i_k=i_{k-1}}^{d} w_{i_1 i_2 \cdots i_k} x_{i_1} x_{i_2} \cdots x_{i_k}), \end{aligned} \tag{1}$$

where $\sigma(\cdot)$ is a suitable (linear or nonlinear) activation function, x_j is the j-th component of input $\mathbf{x} = (x_1, \cdots, x_d) \in \Re^d$, $w_{jkl\cdots} \in \Re$ is an adjustable weight (free parameter) from product of input components $x_j, x_k, x_l, \cdots$ to the output unit, and $w_0 \in \Re$ is the threshold[1]. Since these networks do not have hidden layers, fast learning schemes such as Hebbian or LMS (least mean squares) type rules can be used. The approximation capabilities of this type of networks are obtainable from the Weierstrass polynomial approximation theorem. Unfortunately, the number of weights required to accommodate all higher-order correlations explodes with the input dimension, d and the desired degree of the network, k. In fact, a k-th degree HPU needs a total of

$$\sum_{i=0}^{k} \left(\begin{array}{c} d+i-1 \\ i \end{array} \right) = \left(\begin{array}{c} d+k \\ k \end{array} \right) \tag{2}$$

weights if all products of up to k components are to be incorporated. Consequently, typically only second or third order networks are considered in practice. Such a restriction to the order of the network leads to a reduction in the mapping capability, thereby limiting the use of this kind of higher-order networks.

[1] In this paper, we interchangeably use $\mathbf{x}$ or $(x_1, \cdots, x_d) \in \Re^d$ to denote the input.

List of Figures

Ridge Polynomial Networks

Abstract

This paper presents a polynomial connectionist network called *ridge polynomial network (RPN)* that can uniformly approximate any continuous function on a compact set in multi-dimensional input space $\Re^d$, with arbitrary degree of accuracy. This network provides a more efficient and regular architecture compared to ordinary higher-order feedforward networks while maintaining their fast learning property.

The ridge polynomial network is a generalization of the pi-sigma network and uses a special form of ridge polynomials. It is shown that any multivariate polynomial can be represented in this form, and realized by an RPN. Approximation capability of the RPNs is shown by this representation theorem and the Weierstrass polynomial approximation theorem. The RPN provides a natural mechanism for incremental network growth. Simulation results on a surface fitting problem, the classification of high-dimensional data and the realization of a multivariate polynomial function are given to highlight the capability of the network. In particular, a constructive learning algorithm developed for the network is shown to yield smooth generalization and steady learning.

Ridge Polynomial Networks

Yoan Shin[1] and **Joydeep Ghosh**[2]

[1]Yoan Shin was with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712. He is now with Microelectronics and Computer Technology Corporation (MCC), Austin, TX 78759.

[2]Joydeep Ghosh is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712.