The Pi-sigma Network : An Efficient Higher-Order Neural Network for Pattern Classification and Function Approximation

Yoan Shin and Joydeep Ghosh
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712

Abstract

This paper introduces a novel feedforward network called the pi-sigma network. This network utilizes product cells as the output units to indirectly incorporate the capabilities of higher-order networks while using a fewer number of weights and processing units. The network has a regular structure, exhibits much faster learning, and is amenable to the incremental addition of units to attain a desired level of complexity. Simluation results show good convergence properties and accuracy for function approximation. Comparative results using the DARPA acoustic transient data set are also provided to highlight the classification abilities of pi-sigma networks.

I. Higher-order feedforward networks

Multi-layered perceptron (MLP) networks using the backpropagation learning rule or its variants have been successfully applied to applications involving pattern classification and function approximation. Unfortunately, the training speeds for multilayered networks are extremely slower than those for feedforward networks comprising of a single layer of threshold logic units, and using the perceptron, ADALINE or Hebbian type learning rules [1,2]. Moreover, these networks converge very slowly in typical situations dealing with complex and nonlinear problems, and do not scale well with problem size [2,3].

Higher-order correlations among the input components can be used to construct a higher-order network to yield a nonlinear discriminant function using only a single layer of cells [4]. The building block of such networks in the higher-order processing unit (HPU), defined as a neural processing unit that includes higher-order input correlations, and whose output, y, is given by [5,6]:

$$y = \sigma\Big(\sum_{j} w_j x_j + \sum_{j,k} w_{jk} x_j x_k + \sum_{j,k,l} w_{jkl} x_j x_k x_l + \dots\Big) \; ; \rightarrow$$

[illegible] parameter ใน polynomial

where $\sigma(x)$ is a nonlinear function of input $\boldsymbol{x}$, x_j is the j-th component of $\boldsymbol{x}$, and $w_{jkl\dots}$ is an adjustable weight from product of inputs $x_j, x_k, x_l, \dots$ to the HPU. If the input is of dimension N, then a k-th order HPU needs a total of

$$\sum_{i=0}^{k} \binom{N+i-1}{i}$$

3 7 5

weights if all products of up to $k \le N$ components are to be incorporated [7].

A single layer of HPU (SLHPU) network is one in which only one layer of mapping from input to output using HPUs is considered. The order of an SLHPU network is the highest order of any of the constituent HPUs. Thus output of the k-th order SLHPU is a nonlinear function of up to the k-th order polynomials. Since it does not have hidden units as in sigma-pi network, reliable single layer learning rules such as perceptron, ADALINE, or Hebbian type rules can be used. However, to accomodate all higher-order correlations, the number of weights required increases combinatorially in the dimensionality of the inputs[7]. Limiting the order of the network leads to a reduction in the classification capability of the network.

There have been many approaches which maintain the powerful discrimination capability of higher-order networks while reducing higher-order terms. For example, sigma-pi networks use a hidden layer of higher-order TLUs [2]. A multi-layering strategy using sigma-pi units retains the full capability of a higher-order network using a smaller number of weights and processing units, but its learning speed is slower due to layering. Another approach is to use *a priori* information to remove the terms which are irelevant to the problem in a single layer of higher-order TLUs [4,8,9]. However since it is often difficult to find the properties of input pattern space *a priori*, this strategy has limited applications.

In this paper, we propose a new higher-order network called the pi-sigma network. The motivation is to develop a systematic method for maintaining the fast learning property of single-layer higher-order networks while avoiding the exponential increase in the number of weights and processing units required. We introduce the pi-sigma network in section II and propose its learning rule in section III. Simulation results and conclusions are given in sections IV and V.

II. The pi-sigma network

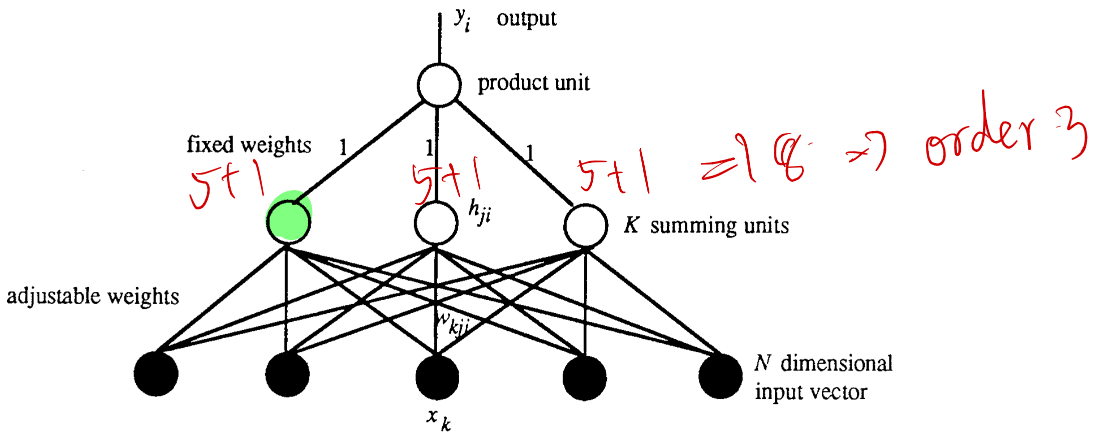

Figure 1. A pi-sigma network.

Fig. 1 shows the proposed *pi-sigma network*. The input $\boldsymbol{x}$ is an N dimensional vector and x_k is the k-th component of $\boldsymbol{x}$. The inputs are weighted and fed to a layer of K linear summing units, where K is the desired order of the network. Let h_{ji} be the output of the j-th summing unit for the i-th output, y_i. Then,

$$h_{ji} = \sum_k w_{kji} x_k + \theta_{ji}, \quad and \quad y_i = \sigma\Big(\prod_j h_{ji}\Big),$$

where w_{kji} is an adjustable weight from input x_k to the j-th summing unit of the i-th output, and θ_{ji} is an adjustable threshold of the j-th summing unit of the i-th output. $\sigma(x)$ denotes the nonlinear activation function, and is selected as the logistic function, $\sigma(x) = 1/(1 + e^{-x})$ for all the results reported in this paper. For binary outputs, the signum or thresholding function can be used. Note that connections from summing units to an output have fixed weights. Thus there is no notion of hidden units in the network and fast learning rules can be used.

The basic idea behind the network is that we can represent the input of a K-th order processing unit by a product of K linear combinations of the input components. That is why this network is called pi-sigma instead of sigma-pi. In the SLHPU representation, the polynomial is represented by summation of all partial products of input components up to order K, thereby leading to an exponential increase in the number of adjustable weights required, as indicated in Table 1. From the network topology point of view, this leads to an irregular structure. In the case of the pi-sigma network, using an additional summing unit increases the network's order by 1 while preserving old connections and maintaining network topology.

order of network	# of weights			
	Pi-sigma		SLHPU	
	N=5	N=10	N=5	N=10
2	12	22	21	66
3	18	33	56	286
4	24	44	126	901

Table 1. Number of weights required for pi-sigma network and SLHPU

III. A Probabilistic learning rule

There are the total of the $(N+1)K$ adjustable weights and thresholds for each output unit, since there are $N+1$ weights associated with each summing unit. The learning rule is a randomized version of the gradient descent procedure. Since the output y_i is a function of the product of all the h_{ji} s, we do not have to adjust all the variable weights at each learning cycle. Instead, at each update step, we *randomly* select a summing unit and update the set of $N + 1$ weights associated with its inputs based on a gradient descent approach. Let

$$e^2 = \frac{1}{2}\sum_p \sum_i (d_i^p - y_i^p)^2 ,$$

where superscript p denotes the p-th training pattern, d_i and y_i are the i-th components of desired and actual outputs, respectively, and the summation is over all outputs and all training patterns. Applying gradient descent on this estimate of the mean square error, we have:

$$\Delta w_{kji} \, \alpha \, -\frac{\partial e^2}{\partial w_{kji}}, \qquad \Delta \theta_{ji} \, \alpha \, -\frac{\partial e^2}{\partial \theta_{ji}} .$$

This yields the update rules for weights and threshold for each iteration step to be:

$$\Delta \theta_{li} = \eta \, (d_i - y_i) y_i' \prod_{j \neq l} h_{ji} ,$$

$$\Delta w_{kli} = \eta \, (d_i - y_i) y_i' \prod_{j \neq l} h_{ji} \, x_k \; = \Delta \theta_{li} \, x_k ,$$

where η is the scaling factor or the learning rate, and y_i' is the first derivative of logistic function $\sigma(x)$, that is, $y_i' = \sigma'(x) = (1-\sigma(x))\sigma(x)$. We reiterate that these updates are applied only to the set of weights and threshold corresponding to the l-th summing unit which is chosen randomly at each step.

IV. Simulation results

To investigate the classification, interpolation and generalization capabilities of the pi-sigma network, we have trained the network to approximate step, rectangular and sinusoidal functions and to solve the classical parity check problem for different input sizes. The performance was compared to that of an SLHPU network of order up to 4 using a gradient descent on the same error measure (e^2) for weight adaptation. The network has also been successfully used to classify underwater SONAR transient data from the DARPA data set 1, and has been observed to require at least an order of magnitude less training time as compared to LVQ and radial basis function classifiers while yielding slightly better results. All simulation codes were written in C++ and run on a Sun 3/80. In the following subsections, we summarize the simulation results.

A. Function approximation

Two types of functions were investigated. The step and rectangular functions and the parity check problem are two-class functions. Fig. 2 shows the result of approximating the rectangular function, $r(x) = 1$ if $-0.4 \leq x \leq 0.4$ and 0 otherwise. Ten evenly spaced values between -1 and 1 were used for training. The cutoff MSE is 0.01 and learning rate, η is 1.0. The number of epochs required are 145 and 237 for networks using 3 and 4 summing units, respectively, where an epoch is defined as one presentation of all training patterns chosen in random order. Since we are interested in approximation rather than classification, the logistic activation function is used for the output unit rather than a thresholding function. For the same cutoff MSE, a third order SLHPU takes 3228 epochs. Interestingly, even with a large value of η, the learning is quite stable, with almost no oscillations being observed in the weight values.

For approximating the unit step function, the pi-sigma network took 107 epochs for a CPU time of 13.2 sec (3 summing units) as compared to a third-order SLHPU which took 3102 epochs and 290.2 secs of CPU time for the same MSE (0.01). The parity check problem is learnt by the pi-sigma network using the same number of summing units as the number of inputs, in time two to three orders of magnitude less than a fine-tuned backpropagation algorithm.

We also trained the network to approximate sinusoidal functions which are not polynomial of inputs, and are not two-class problems. Table 2 highlights the relative performance of the pi-sigma network and SLHPU for the function $f(x) = \cos(\pi x/2)$. Ten training inputs evenly spaced between -1 and

1 are used, and a range of cutoff MSEs and learning rates were experimented with. For 2 summing units, a value of $\eta > 2.0$ resulted in oscillation in error measure and longer times for convergence. When more summing units were used, training was slower than that using 2 summing units for the same value of η. This was mainly due to overfitting of the function, resulting in oscillations in the error measure. Also a lower value of η gives the best results as the number of summing units are increased: an observation made for all the approximation problems that were studied. We also notice that the pi-sigma network is faster than an SLHPU of the same order.

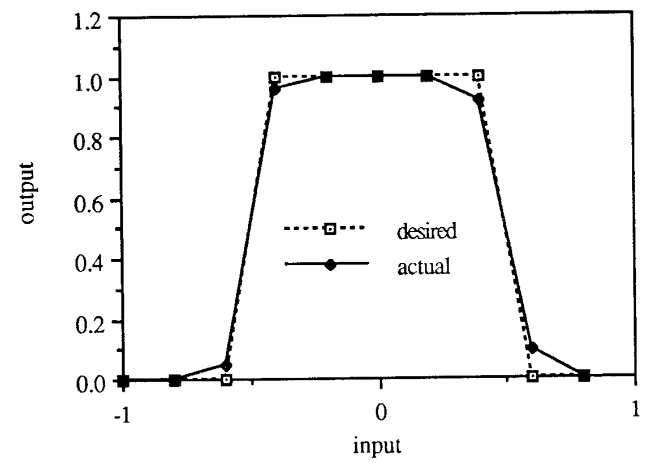

Figure 2. Approximation of rectangular function using pi-sigma network : $\eta = 1.0$, cutoff MSE = 0.01, number of summing units = 3.

	order of network (# of summing units)	learning rate, η	epochs	
			SLHPU	Pi-sigma
cutoff MSE =.015	2	0.1	996	551
		0.5	200	93
		0.75	131	81
		1.0	98	66
		1.5	64	38
		2.0	46	24
	3	0.5	202	170
		0.75	132	73
		1.0	101	76
	4	0.5	891	417
		0.75	297	254
		1.0	314	185

Table 2. Comparison of Simulation results for cosine function approximation using pi-sigma network and SLHPU over a range of learning rates.

Table 3 highlights the scaling of the learning period with the degree of accuracy desired, which is seen to occur in a fairly uniform manner. Fig. 3 shows the actual function realized for different error cutoff values. Similar results are obtained when approximating the sine function, $f(x) = \sin(\pi x)$.

		cutoff MSE	epochs	CPU time(sec)
# of summing units = 2	learning rate, $\eta = 1.0$	0.1	32	3.7
		0.05	45	5.7
		0.03	50	5.9
		0.015	66	7.5
		0.01	89	11.3

Table 3. Simulation results for cosine function approximation using pi-sigma network with different cutoff levels for the MSE.

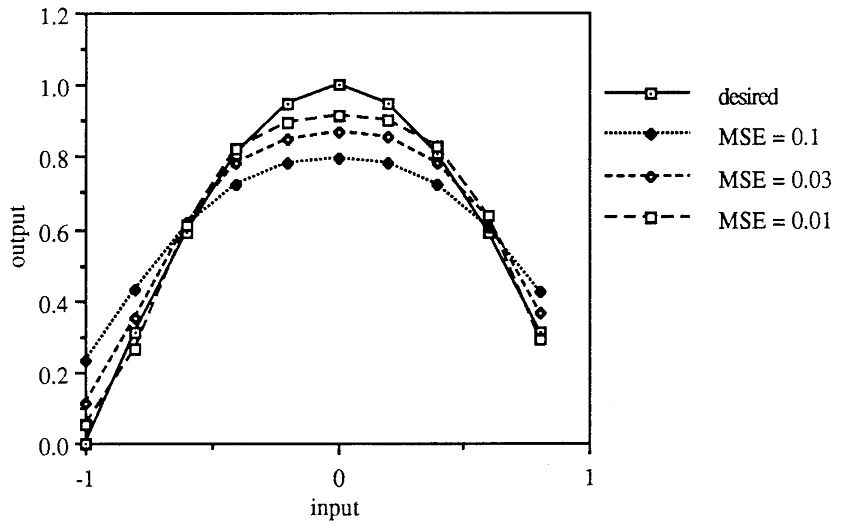

Figure 3. Approximation of cosine function using a pi-sigma network, for varying cutoff MSE levels. Number of summing units = 2 and η = 1.0.

2. Classification of underwater trasient signals

To verify its classification capability for real applications, the pi-sigma network was trained to classify underwater SONAR transient signals taken from the DARPA data set 1. This set has 6 signal types, of which four categories comprise of signals that are of duration between 10 and 100 msecs. The other two categories comprise of longer duration signals and were not considered in this experiment.

Each signal is represented by a 11 dimensional input vector whose components are the AR (autoregressive) filter coefficients. All components are normalized to values between -1 and 1. There are 31 input training patterns and 185 input test patterns belonging to the four categories, representing different underwater sound sources. A 4 dimensional output vector is used. Each component in a desired output vector is either 0 or 1 according to the class the corresponding input vector belongs to. The network was trained using all 31 training patterns. With 4 summing units and with η = 0.75, all training vectors could be classified correctly in 119 epochs.

After training, all weight values are fixed and used to calculate actual output vectors for the 185 test patterns. The maximum component in each output vector indicate the class of the input signal. We obtained 175/185 or 94.6% of classification accuracy for the test patterns. Figure 4 shows the confusion matrix for the 10 misclassifications. These results are better than those reported in [10] using LVQ, radial basis function and multilayerperceptron based classifiers for the same set of inputs and outputs. Moreover, in all the cases of misclassification, the correct class corresponds to the second largest component in the output vector, and has a value close to the largest component (selected class). This suggests that a knowledge-based system can be used in conjunction with the pi-sigma network and additional information to make a final decision for the "outlying" signals.

		misclassified classes			
		1	2	3	4
original classes	1				2
	2				
	3				4
	4	1		3	

Figure 4. Confusion matrix for the 10 misclassified signals from the transient data set.

V. Concluding remarks

The simulation results indicate that fairly complex approximation and classification problems can be tackled by the pi-sigma network using only three or four summing units. This, coupled with the property of stable learning even with values of η over 1, and the modification of only a subset of weights in each epoch, results in a network with significantly reduced training times. We observe that summing units can be added incrementally till an appropriate order of the network is attained without overfitting of the function.

Although we only report results using a constant η, various monotonically decreasing functions of η such as $\eta(t) = a\, e^{-\alpha t}$ and $\eta(t) = c/t$, where t is the epoch step and a, α, c are constants, can be used for more accurate learning. This also allows a larger initial value for the learning rate. Similarly addition of a momentum term and a more selective choice of summing units to be updated can result in further improvements.

Acknowledgements

This research was supported by grant # N00014-89-C-0298 with Dr. Thomas McKenna (ONR) and Dr. Barbara Yoon (DARPA) as government cognizants. We also thank James Whiteley, Larry Deuser and Steven Beck of Tracor Applied Sciences for providing us with preprocessed transient signals, with classification results using several classical as well as neural network techniques, and for several stimulating discussions. The reader can reach Dr. Ghosh at ghosh@ece.utexas.edu for any questions or comments.

References

[1] R. P. Lippmann, " An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp. 4 - 22, April, 1987.
[2] J. McClelland and D. Rumelhart , *Parallel Distributed Processing Vol.1*, The MIT Press, 1987.
[3] B. Widrow and M. A. Lehr, "30 years of Adaptive Neural Networks: Perceptron, Madaline and Backpropagation," *Proc. IEEE*, 78(9): 1415-1422, Sept. 1990.
[4] C. Lee and T. Maxwell, " Learning, Invariance, and Generalization in High-Order Neural Network ," *Applied Optics*, Vol. 26, No. 23, 1987.
[5] R. P. Lippmann, " Pattern Classification using Neural Networks ," *IEEE Commun-ications Magazine*, pp. 47 - 64, 1989.
[6] T. Poggio, " On Optimal Nonlinear Associative Recall ," *Biological Cybernetics 19*, pp. 201 - 209, 1975.
[7] M. Minsky and S. Papert, *Perceptrons*, The MIT Press, 1969.
[8] M. B. Reid et al., " Rapid Training of Higher-order Neural Networks for Invariant Pattern Recognition ," *Proceedings of IJCNN Vol. I*, Washington D. C. , pp. 689 - 692, 1989.
[9] L. Spirkovska and M. B. Reid, " Connectivity Strategies for Higher-order Neural Networks applied to Pattern Recognition," *Proceedings of IJCNN Vol. I*, San Diego, pp. 21 - 26, 1990.
[10] J. Ghosh, L. Deuser and S. Beck, " Impact of Feature Vector Selection on Static Classification of Acoustic Transient Signals," Govt. Neural Network Applications Workshop, San Diego, August 29-31, 1990.