

**EFFICIENT NONLINEAR MAPPING USING THE
MULTI-LAYER PERCEPTRON**

The members of the committee approve the masters
thesis of Arunachalam Gopalakrishnan

Michael T. Manry
Supervising Professor

Jack Fitzer

David P. Klemer

DEDICATION

To my parents

**EFFICIENT NONLINEAR MAPPING USING THE
MULTI-LAYER PERCEPTRON**

by

ARUNACHALAM GOPALAKRISHNAN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 1994

ACKNOWLEDGEMENTS

This research would not had been possible but for the guidance and support of Dr. Michael T. Manry. His continued encouragement, willingness to listen and co-operate in an informal manner go beyond words. I owe sincere thanks to him. Special thanks go to other committee members Dr. Jack Fitzner and Dr. David P. Klemer for reviewing my work. I thank the past and present members of the Image processing and Neural Networks laboratory for providing a friendly atmosphere to work.

I thank my father and mother for their support and encouragement throughout my career. I am deeply indebted to them. I thank my brothers and sister who had been a constant source of inspiration from miles away throughout my graduate program. Also, I thank my friends for the intangible help they had contributed towards the completion of this work.

December 9, 1993

ABSTRACT

EFFICIENT NONLINEAR MAPPING USING THE MULTI-LAYER PERCEPTRON

Publication No. _____

Arunachalam Gopalakrishnan, M.S.

The University of Texas at Arlington, 1994

Supervising Professor: Michael T. Manry

In this thesis, the mapping capability of the Gabor polynomial is reviewed and the nonlinear mapping capability of Multi-layer perceptron(MLP) is discussed in terms of the number of parameters used to represent them. It is shown that the pattern storage capability of multi-variate polynomial is much higher than the commonly used lower bound on multi-layer perceptron pattern storage.

It is also shown that multi-layer perceptron networks having second and third degree polynomial activations can be constructed which implement Gabor polynomials and therefore have the same high pattern storage capability. The polynomial networks are mapped to a conventional sigmoidal MLP resulting in highly efficient network.

It is shown clearly that albeit techniques like output weight optimization and

conjugate gradient algorithm prove to be better than back propagation, as they attain only the lower bound of pattern storage, certainly they are not the final solutions to the training problem. The proposed mapping method attains the upper bound of pattern storage closely.

TABLE OF CONTENTS

Acknowledgements	iv
Abstract	v
List of Figures	ix
List of Tables	x
1. Introduction	1
1.1. Mapping Neural Networks	2
1.2. Scope of the thesis	3
2. Review of Polynomial Mapping Networks	5
2.1. Multi-variate polynomial	5
2.2. Pattern storage of a polynomial	7
3. Review of Multi-Layer Perceptron Mapping Networks	8
3.1. PBF model of multi-layer perceptron	9
3.2. Pattern storage bounds	9
3.3. Basic Problems	12
3.4. Thesis goals	13

4.	Efficient Mapping using Polynomial Perceptron	14
4.1.	Mapping of an arbitrary second degree polynomial	15
4.2.	Cubic polynomial mapping	18
4.3.	3-rd degree polynomial mapping	21
4.4.	Higher degree mapping	26
5.	Efficient Sigmoidal Subnet Design	28
5.1.	Theoretical development of the method	28
5.2.	Practical design procedure	32
6.	Efficient Sigmoidal Networks	35
6.1.	Sigmoidal subnet: n-input, 1-output case	35
6.2.	Networks with multiple outputs	37
6.3.	Design procedure for large networks	40
7.	Examples	42
7.1.	Realizing a 3-rd degree polynomial	42
7.2.	Training algorithms and pattern storage	43
7.3.	Network initialization	45
8.	Conclusions	48
	Appendix	50
	References	60

LIST OF ILLUSTRATIONS

4.1.	Quadratic mapping network	15
4.2.	Cubic mapping network	19
4.3.	3-rd degree function mapping network	23
5.1.	Sigmoidal subnet	29
5.2.	$f^{(2)}(\text{net})/f^{(3)}(\text{net})$ versus net for a sigmoid	30
6.1.	Sigmoidal net with N-inputs and 1 output	36
6.2.	Network with 1 input and many outputs	38
7.1.	Realizing a 3-rd degree polynomial with one hidden unit	42
7.2.	Training error vs. number of patterns	44

LIST OF TABLES

4.1.	Pattern storage in a MLP network for a second degree polynomial	18
4.2.	Pattern storage of a MLP network for a 3-rd degree polynomial	26
6.1.	Number of hidden units as a function of number of inputs and outputs	39
7.1.	Mapping MSE of power load prediction data	46

CHAPTER 1

INTRODUCTION

While simple linear systems are useful in a large number of applications, there are many practical situations that require a nonlinear mapping of the input variables to the output domain. Multi-dimensional nonlinear mapping forms the basis for many prediction, estimation, control and signal processing problems. Multi-variate polynomials or Volterra series expansion can model a large class of nonlinear systems[1-5]. The inherent advantage of Volterra series expansions is that the expansion is a linear combination of nonlinear function of the inputs. Therefore, such polynomials can be designed by solving a system of linear equations. Also, such multi-variate polynomials are attractive because of their ability to approximate many nonlinear systems with great parsimony in the number of coefficients used.

With the advent of the back propagation training algorithm[6-8] there has been a great amount of attention paid to the use of neural networks for multi-dimensional nonlinear mapping. Researchers have shown that a Multi-Layer Perceptron(MLP) type of neural network can approximate an arbitrary nonlinear function in many variables[9-11]. The MLP is being widely used in system identification and nonlinear modeling[12-14]. However, the number of parameters, or weights in neural network terminology, required to characterize a neural network is relatively large. Here, let us investigate this aspect, the related issues and

the implications in design and training of a MLP.

1.1. Mapping Neural Networks

Neural networks estimate functions. MLP can map an input pattern to an output pattern and is a universal approximator. For these reasons, they are called mapping neural networks. We can define a mapping neural network as follows.

A mapping neural network is a distributed parallel network performing a mapping $\phi: I^n \rightarrow R^m$, based on the inter connection of neurons as basic nonlinear computational elements, where I^n is the n -dimensional input space and R^m is the m -dimensional output space.

Several different forms of mapping networks have been reported. The MLP which is the most widely used network, approximately realizes an arbitrary input-output function or a decision function. In the MLP, during a series of intermediate mappings, the input domain is transformed to a complex output space[15]. Kohonen's self organizing feature maps[16,17], Grossberg's network[18], Hecht-Nielson's network[19] and Radial Basis Functions networks[20,21] are some of the different mapping networks which have features of their own. The bidirectional associative memory[22] is a mapping network which is useful for associative information recovery. The Boltzman machine is based on the same principle, but in a probabilistic sense[23].

Mapping neural networks find applications in varied fields. They are model-free estimation systems[24] which model a system without accurate knowledge of the governing equations or parameters. In places where the classical PID control fails because of changes in the system parameters, mapping networks seem to be an attractive alternative[25,26]. In

robotics they can be used in inverse-kinematics, trajectory planning or motion control[27,28]. Nonlinear estimation is another field where a mapping network has potential applications. They can be used as system identifiers[29]. They could be used as demodulators in communication systems[30].

1.2. Scope of the thesis

The mathematical theories behind MLP can be traced back to the so called "13-th problem of Hilbert"[31]. The problem was answered by Kolmogorov[32] and its relevance to the MLP was established by Hecht-Nielsosn[33]. Cybenko[11] showed that it is possible to approximate any multi-dimensional function to any desirable accuracy by superposition of sigmoidal function. He showed that a network with one hidden layer is sufficient to represent any $I^n \rightarrow R$ mapping to an arbitrary accuracy. This forms the theoretical ground for the mapping capability of the multi-layer perceptron[34]. The mapping efficiency of such a network can be defined either in terms of the number of hidden units used, in terms of the number of weights in the network or in terms of the number of patterns the network can store. The number of weights or parameters required in a network to realize an arbitrary function is a subject of concern, as is the pattern storage capability of the network. These two problems are interrelated. Here we discuss the approximation capabilities and the efficiency of a three layer mapping neural network with polynomial or sigmoidal activation, in terms of the number of weights and the pattern storage. Some researchers have looked at the lower bound of the pattern storage[35,36]. Mu-Song Chen and Michael T. Manry[37-39] have modeled MLP using polynomial basis functions. In this thesis, polynomial basis functions are

used to decide on the bounds on the pattern storage capability of the MLP.

The rest of the work is organized as follows. Chapter 2 establishes the bounds on the pattern storage of the multi-variate polynomial. Chapter 3 arrives at the bounds on the pattern storage in a MLP. An efficient mapping technique to design a second and third degree polynomial network is given in chapter 4. Chapter 5 shows a method to construct efficient sigmoidal subnets. Chapter 6 discusses building large sigmoidal networks based on the subnets and the polynomial perceptron. Chapter 7 demonstrates the potential applications of the research through examples. The words *weight* and *parameter* are used interchangeably in the discussions.

CHAPTER 2

REVIEW OF POLYNOMIAL MAPPING NETWORKS

Polynomial mapping networks can be treated as multi-variate polynomial approximations. In comparison to the well developed univariate approximation theory of polynomials the multi-variate theory exhibits a number of distinctive features. If f is a real valued function in R^N , Y is a N -dimensional vector space and $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ are given, then the property that makes it possible to determine $\mathbf{y} \in Y$ which interpolates f at the given points for all functions f and all choices of distinct $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, is called the *Haar* property. In general, polynomials in several variables do not enjoy the *Haar* property which is essentially a univariate concept[42]. Also, there are some other distinguishing features. Univariate polynomials can be used as building blocks in multi-variate approximation by combining them in various ways through addition and multiplication[43]. The basis stems from the theorems like that of Stone-Weierstrass[44], and Nachbin[45]. In this section, we review multi-variate polynomial networks and some of their properties.

2.1. Multi-variate polynomial

Let there be N inputs x_1, x_2, \dots, x_N which need to be mapped to a function $f(\mathbf{x})$ over an N -dimensional space. An N -ary Gabor polynomial for approximation can be written as

$$f(\mathbf{x}) = k_0 + k_1 x_1 + k_2 x_2 + \dots + k_{11} x_1^2 + k_{12} x_1 x_2 + \dots + k_{12 \dots n} x_1 x_2 \dots x_n + \dots$$

$$f(\mathbf{x}) = \mathbf{k}^T \cdot \mathbf{X} \quad (2.1)$$

where \mathbf{k} is the coefficient vector and \mathbf{X} is the vector of the cardinal functions of the polynomial. The elements of the vector \mathbf{X} can be formed by the product of one or more inputs. The width L of a Gabor polynomial f is the number of terms or cardinal functions in the polynomial and is given by

$$L = \sum_{k=0}^P \frac{(N+k+1)!}{(N-1)!k!} = \frac{(N+P)!}{N!P!} \quad (2.2)$$

where P is the degree of the polynomial.

Note that in neural network terminology, the Gabor polynomial is a type of functional link network. Pao[46] uses a system of linear equations to obtain the weights in networks. Suppose there are N_v patterns \mathbf{x}_i ($1 \leq i \leq N_v$) with functional values of f_i ($1 \leq i \leq N_v$). Then the interpolating polynomial for the N_v patterns can be obtained by solving the system of linear equations,

$$\mathbf{f} = \underline{\mathbf{X}} \cdot \mathbf{k} \quad (2.3)$$

for \mathbf{k} , where the rows of the N_v by L matrix $\underline{\mathbf{X}}$ are comprised of vectors \mathbf{X}^T at different data points \mathbf{x}_i , \mathbf{k} is the vector of coefficients of the polynomial and \mathbf{f} is the N_v by 1 vector of the function values.

2.2. Pattern Storage of the polynomial

The pattern storage of a conventional or neural network is the number of input-output pairs the network can reproduce without error. Here we discuss the pattern storage capability of the polynomial discriminant or Gabor polynomial, and relate it to the pattern storage of the multi-layer perceptron (MLP).

Lemma 1. *If the N_v patterns $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N_v}$, contain a subset of L patterns which when expressed as cardinal functions of a Gabor polynomial are linearly independent, then $(\underline{\mathbf{X}}^T \underline{\mathbf{X}})^{-1}$ exists ($\underline{\mathbf{X}}$ has full rank) and there exists a unique Gabor polynomial which can store L patterns.*

The solution fits a hyperplane which satisfies the data at all the L patterns. In general the number of patterns a Gabor polynomial can store is equal to its width i.e., inverse of $\underline{\mathbf{X}}$ may exist when it is a square matrix. But for arbitrarily chosen data $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3, \dots, \mathbf{k}_n$ a hyperplane determined by a unique polynomial function that interpolates the data at all the n points may not exist. Compare this to the univariate case where the Vandermonde matrix is non-singular[47]. This may happen when some of the data are linearly dependent forcing $\underline{\mathbf{X}}$ to be either singular or rank deficient. However even linearly dependent data can become independent when expressed as terms of the Gabor polynomial. For instance, $\mathbf{x}_1 = k \mathbf{x}_2$ where k is scalar constant does not always imply that $\mathbf{X}_1 = k \mathbf{X}_2$, where \mathbf{X}_i is a row vector of $\underline{\mathbf{X}}$. The linear dependency problem can be alleviated by the use of orthogonal polynomials.

CHAPTER 3

REVIEW OF MULTI-LAYER PERCEPTRON MAPPING NETWORKS

One important characteristic of MLP neural networks is that training usually involves iterative techniques, such as the well known back-propagation algorithm[8], for instance, which attempts to minimize a transcendental criterion function. Remember that the output layer of a mapping MLP has linear units, in the sense that they do not have a nonlinear activation function. In the output weight optimization algorithm[48], this is taken advantage to obtain the weights of the output layer by solving a linear system of equations. That would provide a quadratically optimal output weights. The hidden layer and input layer weights are obtained through back-propagation.

Use of conventional optimization techniques for neural network training has been proposed by various authors[49-51]. For unconstrained optimization the conjugate gradient algorithm is widely used. Here again a quadratic error function is minimized. Theoretically the algorithm converges in m iteration where m is number of distinct eigen values of the input matrix. All these training algorithms incur a lot of computational time for convergence. When the algorithms converge, they result in poor utilization of the weights to be demonstrated in chapter 7. Thus they result in an inefficient network.

In this chapter, we begin by reviewing a polynomial model of the MLP and review bounds on MLP pattern storage. We then discuss the problems with the MLP and related

thesis goals.

3.1. PBF model of MLP

Mu-song Chen and M.T. Manry have modeled MLP using a polynomial basis. A MLP with N inputs, I output, and N_h hidden units can be modeled using Polynomial Basis Functions [39] as

$$f(\mathbf{x}) = \mathbf{w}_o^T \mathbf{C} \mathbf{X} \quad (3.1)$$

where \mathbf{w}_o is a I by N_h the output weight vector, \mathbf{C} is a $N+N_h+I$ by L matrix whose rows are the coefficients of the approximating polynomial of each hidden unit, \mathbf{X} is an L by I polynomial cardinal function vector and $f(\mathbf{x})$ is the function to be mapped.

Here each hidden unit is assumed to be approximated by a polynomial and L is the width of the highest degree polynomial in the network.

3.2. Pattern storage bounds

The pattern storage capabilities of the MLP have been examined by several researchers including [35,36,52]. It can be explained largely on the basis of the underlying model of the network. Obviously the pattern storage depends on the number of weights in the network. We will arrive at the bounds on the storage capacity of MLP based on the polynomial basis function representation. Let us define a few terms before we actually show the bounds on the pattern storage.

The number of absolute parameters in a network can be defined as the total number of weights or parameters in a network. The efficiency of the network can be defined as the

ratio of number of parameters to the width of the Gabor polynomial whose degree is equal to that of the network.

All the weights in the network may not be independent of each other to be efficient in mapping. The bounds on pattern storage can be explained on the basis of this representation. The upper bound on the pattern storage depends on the number of effective parameters. Similarly the lower bound depends on the number of elements in the \mathbf{w}_o vector.

Lemma 2. *The multi-layer perceptron can store a minimum number of patterns S_L , equal to the number of output weights connecting one output. i.e., the sum of number of hidden units, the number of inputs and the output threshold.*

$$S_L = N + N_h + 1 \quad (3.2)$$

This result is basically the same as that given in [35,36,52].

Proof: If there are N_v training patterns and I output the equation (3.1) for all the patterns can written as

$$\mathbf{f} = (\mathbf{C}\mathbf{X})^T \mathbf{w}_o \quad (3.3)$$

where \mathbf{f} is N_v by I vector whose rows are comprised of $f_i(\mathbf{x})$'s for ($1 \leq i \leq N_v$) and \mathbf{X} is the L by N_v matrix whose columns are comprised of \mathbf{X}_i 's for ($1 \leq i \leq N_v$). In the above equation an unique solution for \mathbf{w}_o exists when $N_v = N_h + N + I$. Usually the number of training patterns N_v is much greater than the number of units N_h in the network. Nonetheless, a solution for \mathbf{w}_o can be obtained for this over determined system which satisfies $N_h + N + I$ patterns. Therefore the network is capable of storing at least that many number patterns equal to the number of elements in \mathbf{w}_o ($=N_h + N + I$). The result applies to the multiple output case too as

there are same number of output weights for each output.

Let N_w denote the number of weights and thresholds in an MLP which are on the paths from any input to the first output.

Lemma 3. *The upper bound S_U , on the pattern storage capacity may be defined as the number of terms associated with the equivalent Gabor polynomial or the width of the polynomial.*

$$S_U = \min(L, N_w) \quad (3.4)$$

proof: The MLP is capable of storing a much larger number of patterns than that determined by the lower bound since it does not account for the weights other than the output weights. However all the weights in the MLP are not independent i.e., the number of effective parameters is lesser than the total number of parameters. It is because if there are N inputs feeding a sigmoidal unit, the approximating polynomial of the unit has greater number of terms than the number of inputs and hence the weights are dependent. In (3.1), if w_o and \mathbf{C} are multiplied, we get

$$f(\mathbf{x}) = \mathbf{y}^T \mathbf{X} \quad (3.5)$$

where \mathbf{y} is a L by 1 vector. Again when we have N_v patterns a unique solution can be found for \mathbf{y} can be found when $N_v = L$. Thus the number of parameters the network can store cannot be greater than the width of the equivalent Gabor polynomial. However, if the number of weights in the path of a input to the output is less than the width of the polynomial then obviously the pattern storage is bounded by that and hence the lemma.

3.3. Basic problems

The actual pattern storage of an MLP is somewhere between that of its lower limit in *lemma 2* and its upper limit in *lemma 3*. If the lower bound in *lemma 2* turns out to be close to the correct value, then the MLP is likely to be superseded in the future by Volterra filters, functional link nets, and other networks which more directly implement Gabor polynomials. Our goal in this paper is to show that MLPs with polynomial or monomial activation can efficiently implement the Gabor polynomial. Since sigmoidal networks can closely approximate polynomial networks [37-39], they also have high pattern storage capability.

As we saw earlier a continuous function in N variables can be approximated to an arbitrary accuracy using Gabor polynomial expansion or using orthogonal expansions. The advantages of neural networks over such conventional techniques is not clear. MLP has proved to have a distinct superiority in performance over other classification techniques. MLP may represent a discriminant functions which are used in classifiers, well but they may not represent an arbitrary mapping efficiently compared to polynomials.

Albeit, it had been shown that a neural network with a single hidden layer with nonlinear activations is capable of mapping to an arbitrary accuracy, those neural networks are not very efficient representation of multi-variate approximation in terms of the number of parameters required to represent them. Multi-layer perceptron is clearly redundant in terms of number of weights or parameters associated with it compared to the number of terms used in an equivalent Gabor polynomial which can represent the same function. As we saw earlier the polynomial can store that many number of patterns equal to its width. Should the MLP

be as efficient, it must be able to store that many number of patterns equal to the number of weights in the network. We know that that is not the case always. Training algorithms attain only the lower bound of pattern storage. Here, we propose some efficient networks whose pattern storage capability is equal to the number of weights in the network.

3.4. Thesis goals

The mapping capabilities of a multi-variate polynomial are established to a certain extent. Here, the mapping capabilities of a single hidden layer perceptron structure with a polynomial or sigmoidal activation and its relevance to a polynomial are investigated. The mapping efficiency and the pattern storage capabilities of the structure are analyzed in terms of the number of parameters associated with it. A constructive proof is given to design a three layer network so as to reach the upper bound on the pattern storage capability and to reach the lower bound on the number of weights or parameters involved. Design techniques are given for building large efficient sigmoidal networks. The implications of the direct mapping on training algorithms and their deficiencies are demonstrated.

Some of the notation convention used in our discussions are as follows.

w_{ij} - weight connecting the i -th neuron to the j -th input.

c_k - output weight connecting the i -th neuron to the output.

k_{ijk} - coefficient of a term $x_i x_j x_k$ in a polynomial.

In our discussions a constant weight of unity is not counted as a parameter for obvious reasons.

CHAPTER 4

EFFICIENT MAPPING USING POLYNOMIAL PERCEPTRON

In this chapter, let us look at polynomial perceptrons for efficient mapping. Polynomial perceptrons are those networks with monomial or polynomial activation function for the hidden units. A special kind of three layer network with sparse connectivity is proposed for mapping a given second degree or third degree polynomial. The following theorem is stated with regard to the quadratic and cubic mapping.

Theorem 1. *If 'p' is any continuous function in [a,b] approximated by a quadratic or cubic polynomial 'f' then the same can be realized with an efficient perceptron structure to an arbitrary accuracy.*

Corollary 1: *If the approximation polynomial consists only of product terms without the second degree or third degree terms of any of the product elements then the function can be approximated by the structure to an arbitrary accuracy such that the error of approximation $\epsilon > 0$.*

Corollary 2: *The mapping can be extended to an arbitrary second or third degree function with a direct connection between the inputs and output which introduces the linear terms and a threshold at the output for the constant term.*

A constructive proof of the above the theorem can be seen in the following sections as such efficient networks are constructed.

4.1. Mapping of an arbitrary second degree polynomial

First, let us look at the quadratic polynomial mapping to a neural network. A sparsely connected network as shown in figure.4.1 is chosen. The network can be described by the following proposition.

A quadratic function of N bounded inputs can be approximated to an arbitrary accuracy by an N -input single hidden layer perceptron structure which has N hidden units with squaring activations where the k -th hidden unit has $N-k+1$ inputs feeding it, where each input is multiplied by a variable parameter (weight) and one input is fed through a weight with unit gain.

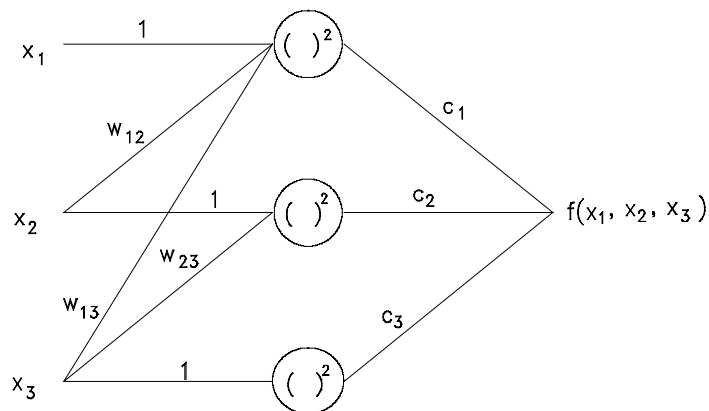


Figure 4.1. Quadratic mapping network

A general quadratic polynomial in N variables can be written as

$$f(x_1, x_2, \dots, x_N) = \sum_{i=1}^N \sum_{j=i}^N k_{ij} x_i x_j \quad (4.1)$$

If this polynomial is mapped on to a network shown in fig. 4.1, then the weights in the network and the polynomial coefficients are related by N equations of the form

$$k_{ii} = c_i + \sum_{j=1}^{i-1} w_{ji}^2 c_j \quad 1 \leq i \leq N \quad (4.2)$$

and $N(N-1)/2$ equations of the form

$$k_{ij} = 2 \sum_{m=1}^j w_{mj} w_{mi} c_m \quad \begin{array}{l} 1 \leq i \leq N \\ i \leq j \leq N \\ i \neq j \end{array} \quad (4.3)$$

As an example let us look at the mapping of a ternary quadratic on to the neural network structure. A general ternary quadratic polynomial can be written as

$$f(x_1, x_2, x_3) = k_{11}x_1^2 + k_{22}x_2^2 + k_{33}x_3^2 + k_{12}x_1x_2 + k_{13}x_1x_3 + k_{23}x_2x_3 \quad (4.4)$$

When this polynomial is mapped on to the network the weights and the coefficients of the polynomial are related by the following set of equations.

$$k_{11} = c_1$$

$$k_{12} = 2w_{12}c_1$$

$$k_{13} = 2w_{13}c_1 \quad (4.5)$$

$$k_{22} = w_{12}^2c_1 + c_2$$

$$k_{23} = 2(w_{12}w_{13} + w_{23}c_2)$$

$$k_{33} = w_{13}^2c_1 + w_{23}^2c_2 + c_3$$

The weights of the network can be obtained easily by solving these equations in the above order.

The solution of the above set of equations may become a problem if one or more terms in the polynomial vanish. If the polynomial consists only of product terms then the equations have to be solved by replacing the coefficients of square terms with a small positive constant ϵ . In this case the mapping error will always be greater than zero abiding corollary 2 to theorem 1. If some of the square terms vanish and some of them exist then the equations can be solved by switching the input with square terms as the first term (as x_j) followed by the inputs without square terms in the network. More generally, an N -ary second degree polynomial can be mapped similarly, with the introduction of direct connection between inputs and outputs for linear terms and an output threshold for the constant term. Algorithm QUAD_NET, given in the appendix solves the equations to obtain a second degree polynomial perceptron.

Table.4.1 shows some results on the pattern storage capacity of the proposed structure as compared to that of the second degree Gabor polynomial for different number of inputs. It can be seen that the proposed structure stores the same number of patterns as the number of weights in the network which also is the upper bound on the pattern storage for a second degree network.

Table 4.1. Pattern storage in a MLP network for a Second degree polynomial

# of inputs N	# hidden units N_h	Pattern storage N_p	Upper bound S_U	Lower bound S_L
2	2	6	6	5
3	3	10	10	7
4	4	15	15	9
5	5	21	21	11
6	6	28	28	13

4.2. Cubic Polynomial Mapping

Another sparsely connected network is proposed for cubic polynomial mapping. An example network for a three input case is given in figure 4.2. The network can be described as follows.

An N-ary cubic polynomial function with bounded parameters can be realized by single hidden layer neural network of N inputs with $N(N+1)/2$ hidden units with cubing activation functions where N units are connected to 1 input, N-1 units are connected to 2 inputs, N-2 units are connected to 3 inputs and so on.

Each hidden unit with multiple inputs have their input weights as variable parameters and their outputs are unity whereas those units with one input have their input weight as unity and their outputs are variables. The number of units N_h used for an N-ary 3-rd degree polynomial is

$$N_h = \frac{N(N+1)}{2} \quad (4.6)$$

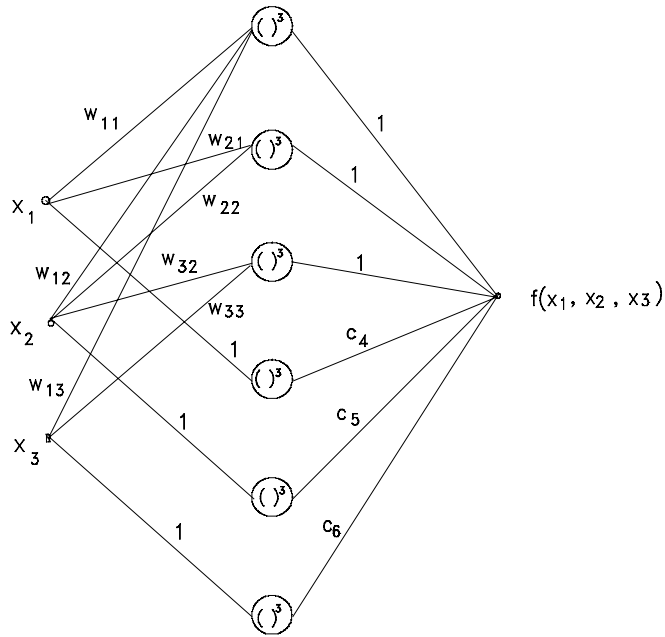


Figure 4.2. Cubic mapping

A general N -ary cubic polynomial can be written as

$$f(x_1, x_2, \dots, x_N) = \sum_{i=1}^N \sum_{j=i}^N \sum_{l=1}^N k_{ijl} x_i x_j x_l \quad (4.7)$$

When this polynomial is mapped on to a structure shown in fig. 4.2 we get the following set of equations. There are N equations of the form

$$k_{iii} = \sum_{j=1}^{N_h} w_{ji}^3 c_j, \quad 1 \leq i \leq N \quad (4.8)$$

$N(N-1)/2$ equations of the form

$$k_{ijj} = 3 \sum_{l=1}^{N_h} w_{li}^2 w_{lj} \quad \begin{array}{l} 1 \leq i \leq N \\ i \leq j \leq N \\ i \neq j \end{array} \quad (4.9)$$

$N(N-1)/2$ equations of the form

$$k_{ijj} = 3 \sum_{l=1}^{N_h} w_{li} w_{lj}^2 \quad \begin{array}{l} 1 \leq i \leq N \\ i \leq j \leq N \\ i \neq j \end{array} \quad (4.10)$$

and $N(N-1)(N-2)/6$ equations of the form

$$k_{ijl} = 6 \sum_{m=1}^{N_h} w_{mi} w_{mj} w_{ml} \quad \begin{array}{l} 1 \leq i \leq N \\ i \leq j \leq N \\ j \leq l \leq N \\ i \neq j \neq l \end{array} \quad (4.11)$$

As in the case of the quadratic mapping an example of a ternary cubic mapping is given. The general polynomial for a ternary cubic function is

$$\begin{aligned} f(x_1, x_2, x_3) = & k_{111}x_1^3 + k_{222}x_2^3 + k_{333}x_3^3 + k_{112}x_1^2x_2 + k_{122}x_1x_2^2 + k_{113}x_1^2x_3 + k_{133}x_1x_3^2 + k_{223}x_2^2x_3 \\ & + k_{233}x_2x_3^2 + k_{123}x_1x_2x_3 \end{aligned} \quad (4.12)$$

This polynomial when mapped on to the structure shown in fig.4.2 results in the following set of equations which can be solved in the same order to obtain the values of weights in the network.

$$k_{113} = 3w_{11}^2w_{13}$$

$$\begin{aligned}
k_{133} &= 3w_{11}w_{13}^2 \\
k_{123} &= 6w_{11}w_{12}w_{13} \\
k_{112} &= 3(w_{11}^2w_{12} + w_{21}^2w_{22}) \\
k_{122} &= 3(w_{11}w_{12}^2 + w_{21}w_{22}^2) \\
k_{223} &= 3(w_{12}^2w_{13} + w_{32}^2w_{33}) \\
k_{233} &= 3(w_{13}^2w_{12} + w_{32}w_{33}^2) \\
k_{111} &= c_4 + w_{21}^3 + w_{11}^3 \\
k_{222} &= c_5 + w_{32}^3 + w_{22}^3 + w_{12}^3 \\
k_{333} &= c_6 + w_{33}^3 + w_{13}^3
\end{aligned} \tag{4.13}$$

The first two equations can be solved for w_{11} and w_{13} . w_{12} can be obtained from the third equation. Then the next two equations can be solved for w_{21} and w_{22} and the procedure can be continued to solve all the equations. Similar to the case of a quadratic network the solution may become a problem when some of the terms vanish and the situation can be remedied following the same techniques. The CUBE_NET algorithm constructs an N-ary cubic network from an N-variable cubic polynomial.

4.3. 3-rd degree polynomial mapping

As we saw in the last section, an arbitrary cubic function can be efficiently mapped on to the neural network structure. The same structure can be extended to represent an

arbitrary 3-rd degree polynomial by replacing the cubic activation function with a polynomial activation.

An arbitrary N -ary 3-rd degree polynomial function can be efficiently represented by a neural network with a single hidden layer with $N(N+1)/2$ units with a polynomial activation function of the form $bx^3 + ax^2$ where N units have one input, $N-1$ units have 2 inputs and so on, with all the inputs tied to the output and with a output threshold.

The net function, the activation function and the output are,

$$net_k = \sum_{i=1}^N w_{ki} x_i \quad (4.14)$$

$$f(net) = a_k net^2 + b_k net^3 \quad (4.15)$$

$$f(\mathbf{x}) = \sum_{i=1}^{N_h} f(net_i) + \sum_{i=1}^N k_i x_i + k_0 \quad (4.16)$$

The values of b_k coefficients are unity for N_h-N units which are connected to more than one input. Thus N units have b_k coefficients, counted as free parameters. All the output weights are unity.

A general N -ary 3-rd degree polynomial can written as

$$f(x_1, x_2, \dots, x_N) = \sum_{i=1}^N \sum_{j=i}^N \sum_{l=j}^N k_{ijl} x_i x_j x_l + \sum_{i=1}^N \sum_{j=i}^N k_{ij} x_i x_j + \sum_{i=1}^N k_i x_i + k_0 \quad (4.17)$$

and when this polynomial is realized by the network in fig. 4.3. the weights in the network and the 3-rd degree monomial coefficients are related by (4.8) - (4.11) with c 's replaced by b 's. The 2-nd degree monomial coefficients and the a 's are related by N equations of the form

$$k_{ii} = \sum_{j=1}^{N_h} w_{ji}^2 a_j \quad 1 \leq i \leq N \quad (4.18)$$

and $N(N-1)/2$ equations of the form

$$k_{ij} = 2 \sum_{l=1}^{N_h} w_{li} w_{lj} a_l \quad \begin{matrix} 1 \leq i \leq N \\ i \leq j \leq N \\ i \neq j \end{matrix} \quad (4.19)$$

where N_h is given by (4.6).

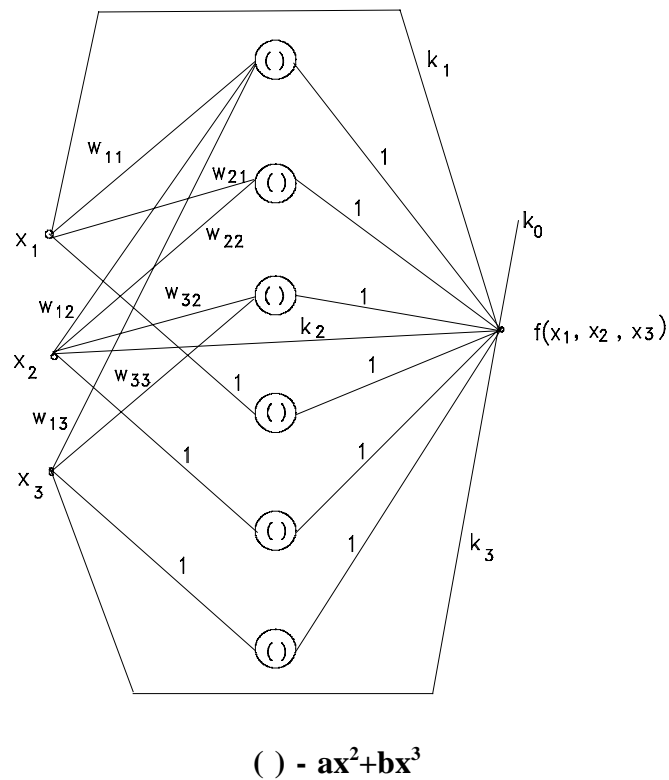


Figure 4.3. 3-rd degree function mapping network

Let us take the example of mapping an arbitrary ternary 3-rd degree Gabor

polynomial. A general ternary 3-*rd* degree Gabor polynomial is

$$\begin{aligned}
 f(x_1, x_2, x_3) = & k_{111}x_1^3 + k_{222}x_2^3 + k_{333}x_3^3 + k_{112}x_1^2x_2 + k_{122}x_1x_2^2 + k_{113}x_1^2x_3 + k_{133}x_1x_3^2 + k_{223}x_2^2x_3 \\
 & + k_{233}x_2x_3^2 + k_{123}x_1x_2x_3 + k_{11}x_1^2 + k_{22}x_2^2 + k_{33}x_3^2 \\
 & + k_{12}x_1x_2 + k_{13}x_1x_3 + k_{23}x_2x_3 + k_1x_1 + k_2x_2 + k_3x_3 + k_0 \quad (4.20)
 \end{aligned}$$

This Gabor polynomial when mapped to the proposed structure results in the following set of equations which when solved in the given order, yields the values of weights.

$$k_{113} = 3w_{11}^2 w_{13}$$

$$k_{133} = 3w_{11} w_{13}^2$$

$$k_{123} = 6w_{11} w_{12} w_{13}$$

$$k_{112} = 3(w_{11}^2 w_{12} + w_{21}^2 w_{22})$$

$$k_{122} = 3(w_{11} w_{12}^2 + w_{21} w_{22}^2)$$

$$k_{223} = 3(w_{12}^2 w_{13} + w_{32}^2 w_{33})$$

$$k_{233} = 3(w_{13}^2 w_{12} + w_{32} w_{33}^2)$$

$$k_{111} = b_4 + w_{21}^3 + w_{11}^3$$

$$k_{222} = b_5 + w_{32}^3 + w_{22}^3 + w_{12}^3 \tag{4.21}$$

$$k_{333} = b_6 + w_{33}^3 + w_{13}^3$$

$$k_{13} = 2w_{11} w_{13} a_1$$

$$k_{12} = 2(w_{11} w_{12} a_1 + w_{21} w_{22} a_2)$$

$$k_{23} = 2(w_{12} w_{13} a_1 + w_{32} w_{33} a_3)$$

$$k_{11} = w_{11}^2 a_1 + w_{21}^2 a_2 + a_4$$

$$k_{22} = w_{11}^2 a_1 + w_{22}^2 a_2 + w_{32}^2 a_3 + a_5$$

$$k_{33} = w_{13}^2 a_1 + w_{33}^2 a_3 + a_6$$

The linear terms and the constant term can be easily added to the equation by connecting the inputs directly to the output with appropriate gains and by introducing a threshold at the output respectively. The complete construction of a third degree network is implemented using the 3D_NET algorithm in the appendix. Some results of the pattern storage capabilities of a third degree Gabor polynomial and that achieved with the proposed neural network structure are tabulated.

All the mapping structures shown above are efficient in terms of number of parameters or weights associated with them. In all the networks it can be seen that the number of weights is equal to the number of constants in the equivalent Gabor polynomial and they are capable of storing equal number of patterns. These are not restricted to any definite number of inputs. Also, the result can be extended to multiple outputs where each output is defined by a different function.

Table 4.2. Pattern storage of MLP network for a third degree polynomial

# of inputs N	# hidden units N _h	Pattern storage N _p	Upper bound S _U	Lower bound S _L
2	3	10	10	5
3	6	20	20	10
4	10	35	35	15
5	15	56	56	21

# of inputs N	# hidden units N_h	Pattern storage N_p	Upper bound S_U	Lower bound S_L
6	21	84	84	28
7	28	120	120	36
8	36	165	165	45

4.4. Higher degree mapping

As the degree of the function is increased one should expect a similar kind of efficient mapping to exist. But surprisingly it does not. As the degree increases beyond three an efficient mapping of an arbitrary polynomial by a neural network structure fails and leads to the following result.

Theorem 2: *An efficient realization of a quartic polynomial using a single layer structure with monomial activation functions is impossible in the sense of number of free parameters associated with it.*

Though this result is apparently startling, it is actually an extension of some of the established facts in geometry and linear algebra. Since there are fifteen parameters in a general ternary quartic and there are three parameters in a ternary form, one should expect a general ternary quartic to be a sum of five fourth powers. Clebsch[53] proved geometrically that a general ternary quartic cannot be written as a sum of five fourth powers. In our terms, fifteen parameters do not suffice to represent any ternary quartic as a sum of powers because the fifteen polynomials in the fifteen variables are not algebraically independent. Sylvester[54,55] reformulated Clebsch's result. In effect he showed that if f is a sum of five

fourth powers, then the 6 by 6 matrix associated to H_p , the *psd* quadratic form has less than full rank and so its determinant, the catalecticant, vanishes.

Thus for mapping a quartic function in a 3-input network we need more than fifteen parameters to represent an arbitrary function. This limitation can be partially overcome by utilizing two hidden layers. The first hidden layer can generate all second and third degree terms in \mathbf{X} fairly efficiently. The second hidden layer would be third degree, resulting in a network with an overall degree of $P=9$. These results indicate why such efficient mapping is impossible for higher degrees. The same can be proved easily in the case of a neural network with single hidden layer. Generalizing, any higher degree (> 3) mapping with a neural network structure with monomial activation will be less efficient than the corresponding Gabor polynomial in the sense of number of parameters required to represent it.

CHAPTER 5

EFFICIENT SIGMOIDAL SUBNET DESIGN

It is clear that MLP networks with polynomial activation function can have efficient pattern storage. In this chapter, let us see how to implement the polynomial hidden unit of (4.5) as a sigmoidal unit and consequently in chapter 6 we will build large sigmoidal networks for efficient pattern storage. Every sigmoidal hidden unit realizes a square and cube term so that it can directly replace the units with polynomial activation.

5.1. Theoretical development of the method

Consider the network in fig. 5.1. The hidden unit has a sigmoidal activation function whose input is the net function,

$$net = w_i x + \theta \quad (5.1)$$

where θ is the threshold of the hidden unit, output is s_o ,

$$s_o = f(net) = \frac{1}{1 + e^{-net}} \quad (5.2)$$

and the network output is

$$p(x) = xw_{o1} + s_o w_{o2} + \theta \quad (5.3)$$

The familiar Taylor series is an efficient tool in function approximation and its

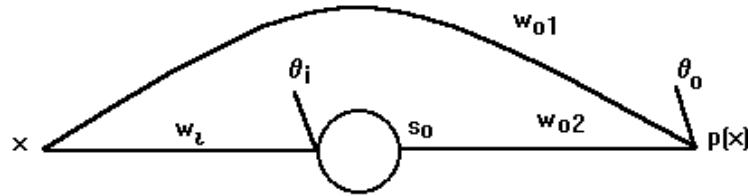


Figure 5.1. Sigmoidal subnet

application[56-58]. Let us consider a function $f(x)$ which is analytic in the neighborhood of a point $x = x_0$. In our case, it is the sigmoidal activation function $(1/1+e^{-x})$ which is analytic about every point. However the analysis is not restricted to sigmoidal activation function.

Within a region of convergence we may write $f(net)$ as

$$f(net) = \sum_{m=0}^n \frac{f^{(m)}(net)}{m!} (net-x_0)^m + R_n(net) \quad (5.4)$$

where $R_n(x)$ is called the remainder. Clearly the series will converge and represent $f(x)$ within the region of convergence. If we put $\theta = x_0$, (5.4) simplifies as

$$f(net) = \sum_{m=0}^n \frac{f^{(m)}(\theta)}{m!} (w_i x)^m + R_n(w_i x) \quad (5.5)$$

The functional approximation accuracy depends upon the number of terms n in the series used to compute the function.

The goal is to realize a third degree polynomial with a single hidden unit with an analytical activation function. In (5.5) let us consider the terms up to third degree and let the polynomial to be mapped be

$$p(x) = bx^3 + ax^2 + cx + d \quad (5.6)$$

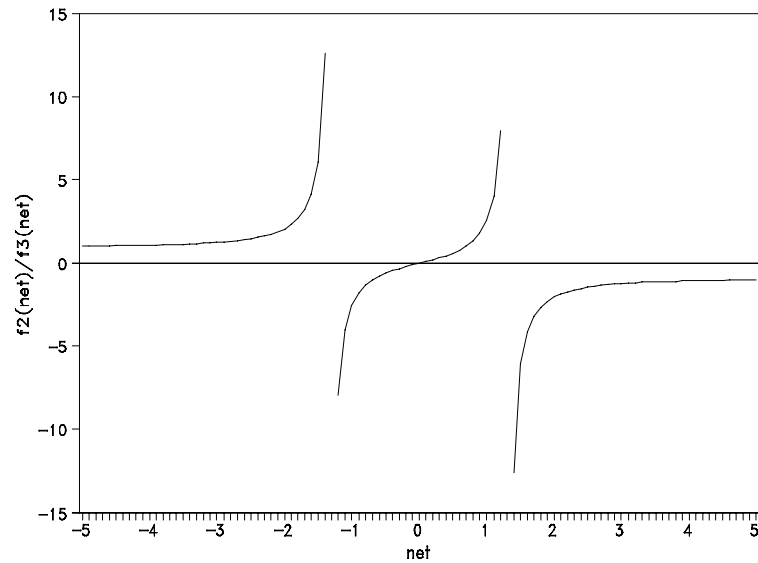


Figure 5.2. $f^2(\text{net})/f^3(\text{net})$ versus net for a sigmoid

The idea is to move the threshold of the hidden unit, θ to a point on the sigmoid curve where it best approximates the second degree and third degree terms in the given polynomial and to correct the errors introduced by the linear term and constant term by the direct connection between the input and output and the output threshold respectively. Note that the ratio of second derivative to third derivative of the sigmoidal function is continuous between negative infinity and positive infinity for $|net| \leq 1.3$ (fig. 5.2). The excursion of the input about the threshold is controlled with the input weight. Let us pick a operating point θ such that the ratio of square term to the cube term in the given polynomial is equal to the

ratio of the coefficients of the second degree term to the third degree term in the series expansion (5.5) of the activation function. i.e., pick θ such that

$$\frac{a}{b} = \frac{3(f''(\theta) - m_x)}{w f'''(\theta)} \quad (5.7)$$

where m_x is the mean of the input signal x . This produces the same ratio of second degree to third degree term as the desired one. By choosing a proper output weight w_{o2} (5.3) we can get the desired second and third degree terms of the polynomial. The error introduced in the linear term ($f'(\theta)w_i w_{o2}$) can be corrected by changing w_{o1} (5.3) and the constant $f(\theta)$ can be eliminated using the output threshold θ_o .

Obviously, the remainder error,

$$R_4(x) = \left[\sum_{m=4}^{\infty} \frac{(w_i x)^m f^{(m)}(\theta)}{m!} \right] w_{o2} \quad (5.8)$$

can not be completely eliminated. A careful examination of the above expression (5.8) gives some information to minimize this error. The error accumulates exponentially with the input weight and the input itself. Therefore one would select an input weight such that the maximum value of the product $w_i x$ to be less than one so that the error would decrease with the higher powers. Similarly we need an activation function whose derivatives are less than one and whose higher derivatives decrease rapidly. It can be seen that the sigmoidal activation function satisfies these requirements. This also leaves us with a possibility to explore other activation functions which might have better approximation capabilities. However that is beyond the scope of this work.

From the above inferences, we need an input weight w_i such that the maximum

excursion of the net function about the operating point is less than one. On the other hand, w_i can not be made arbitrarily small since that would not allow sufficient deviation of the net function about the threshold to produce the polynomial output for different inputs. An optimal region of convergence, R_o has to be selected in between zero and one where the error is within acceptable limits and allowing the necessary deviation. This value can be found through experiments.

Though the output weights can be obtained directly from equations (5.5) and (5.6), when there are more than one data input to take care of the linear error and constant error introduced by different data they are obtained by solving a linear system of equations. In this case we have three unknowns w_{o1} , w_{o2} , and θ_o with inputs from the hidden unit and the from the input itself. The desired polynomial forms the output.

5.2. Practical design procedure.

Refer to the network shown in fig. 5.1. Let the polynomial to be realized be

$$p(x) = bx^3+ax^2+cx+d \quad (5.9)$$

Step 1. Compute the ratio of the coefficient of the square term to the cubic term(a/b) and compute the mean of the input data as

$$m_x = \frac{1}{N_v} \sum_{i=1}^{N_v} x_i \quad (5.10)$$

Step 2. Find the optimal region of convergence R_o for the given ratio of a/b . In our experimental simulations the optimal region of convergence is found to vary between .3 to

.65 for a/b ratio of .001 to 1000 quite linearly. Note that the region of convergence can be obtained off-line and stored for further use. As an alternative, a simple linear relationship can be obtained between the ratio(a/b) and the optimal region of convergence.

Step 3. Using the optimal region of convergence and the maximum and minimum values of the input data compute the input weight as

$$w_i = \frac{2R_o}{x_{\max} - x_{\min}} \quad (5.11)$$

where x_{\max} and x_{\min} are the maximum and minimum values of the input respectively.

Step 4. Obtain a point on the activation function which approximates the given ratio of square to cube term the best. In our implementation it is obtained using Newton-Raphson algorithm with a minor modification added to it. If $f(x)$ is the activation function,

1. Compute $r(x)$ as

$$r(x) = \frac{f''(x) - m}{f'''(x)} - \frac{w_i a}{3b} \quad (5.12)$$

2. Find x such that

$$x_{n+1} = x_n - \frac{r(x_n)}{r'(x_n)} \alpha \quad (5.13)$$

Repeat 2 until the error is within limits. An error threshold of 1×10^{-16} is used in the experiments.

The learning factor α helps in two ways.

(i) Since the ratio $r(x)$ is a monotonically increasing function in both positive and negative directions we can start the iterations with a initial guess of zero always. This

eliminates the need for additional computation for initial guess.

(ii) Also when $r(x) > 1$, α brings down the amount excursion to a new point on the curve and thus guarantees convergence.

This value x_n , after sufficient number of iterations when the required error limit is reached, makes the threshold for the hidden unit as

$$\theta_i = x_n - w_i m \quad (5.14)$$

Step 5. Up to this point all the input weights are available. To obtain the output weights a system of linear equations is solved. The output weights are obtained by minimizing the mean square error

$$\sum_{i=1}^{N_p} (w_{o1} x_i + w_{o2} s_o + \theta_o - p(x_i))^2 \quad (5.15)$$

That forms a system of linear equations in the w_o 's as the unknowns. The input, output of the hidden unit and the desired polynomial output are written as inputs and output and solved to obtain w_{o1} , w_{o2} and θ_o . We used conjugate gradient algorithm[59-61] to solve this least squares problem. That completes the mapping of a 3-rd degree polynomial unit on to a sigmoidal unit as all the weights of the network are available.

CHAPTER 6

EFFICIENT SIGMOIDAL NETWORKS

The subnet designed in the last chapter can be used to build large networks. The mapping method can be extended to units with multiple inputs as basically the method maps the polynomial on to sigmoid based on the net function rather than on the inputs. The procedure given in section 5.2 can be modified for the N -input case. But the region of convergence gets divided among the different inputs depending upon their range. Similarly the sum of the means of all the inputs needs to be subtracted from the value x_n obtained from the Newton's method to get the threshold of the hidden unit.

6.1. Sigmoidal subnet: n -input, l -output case

Refer to fig.6.1 for a sigmoidal subnet with N inputs and l output. Compute the optimum value of deviation D_2 as

$$D = \sum_{i=1}^N \max(x_i) - \min(x_i) \quad (6.1)$$

$$D_2 = \frac{2R_o}{D} \quad (6.2)$$

where $\max(x_i)$ and $\min(x_i)$ are the maximum and minimum values of the i -th input signal x_i respectively. Section 5.2. gives R_o .

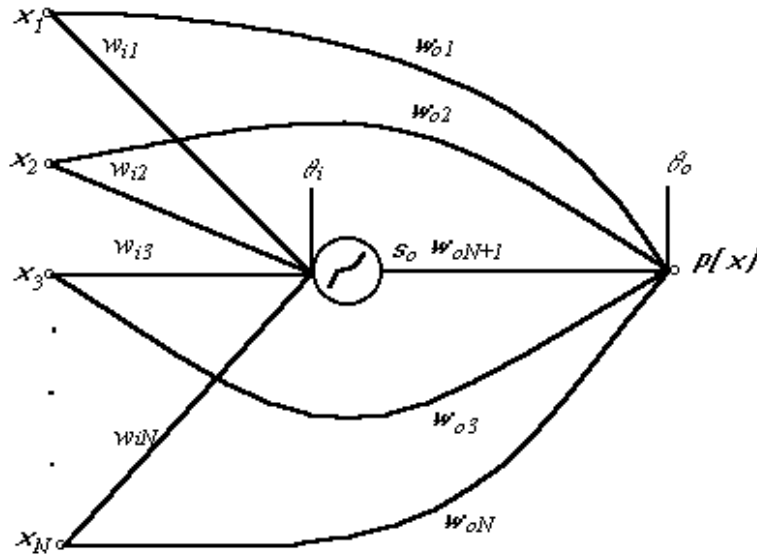


Figure 6.1. Sigmoidal net with N-inputs and 1 output

Find the input weight of the k -th unit w_{ik} as

$$w_{ik} = \frac{D_2(\max(x_k) - \min(x_k))}{D} \quad (6.3)$$

where D and D_2 are from (6.1) and (6.2). Obtain the threshold θ using (5.11), (5.12) and (5.13) where w_i is replaced by D_2 of (6.2). The mean m is the sum of individual means of the inputs given by

$$m_x = \sum_{i=1}^N m_x(i) \quad (6.4)$$

where $m_x(i)$ is the mean of the i -th input. The output weights w_{oi} ($1 \leq i \leq N+1$) and the output threshold θ_o can be obtained by solving the linear system of equations

$$\sum_{i=1}^N x_{ji} w_{oi} + s_o w_{oN+1} + \theta_o = p(x_j) \quad (6.5)$$

for $(1 \leq j \leq N_v)$.

6.2. Networks with multiple outputs

So far we have been considering only networks with one output. Often we come require networks with more than one output, for instance, estimating two parameters from a given data. The mapping method can be extended to the case of multiple inputs-multiple outputs.

To illustrate the design and the hidden unit requirements for such a general network let us begin with a case of 1 input and many outputs(fig. 6.2).

We know that we need one hidden unit to realize a 1 -input 1 -output 3 -rd degree network. It is obvious that to get a second output O_2 , independent of the first output O_1 , we need an additional unit. Note that it is possible to generate a second output with a monomial activation, but anyway we need an additional unit.

The third output O_3 can be generated using these two units by choosing proper output weights. Let us say that the unit 1 has an activation function

$$a_1 x^2 + b_1 x^3 \quad (6.6)$$

and the second unit has activation function

$$a_2 x^2 + b_2 x^3 \quad (6.7)$$

If the polynomial to be realized by the third activation is

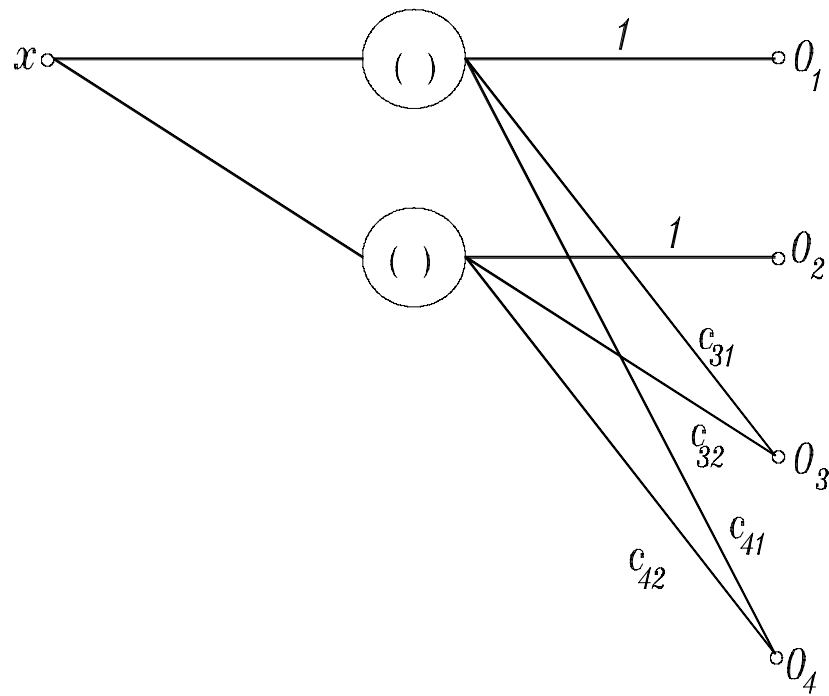


Figure 6.2. Network with 1 input and many outputs

$$a_3x^2 + b_3x^3 \quad (6.8)$$

then it can be obtained by solving the system of linear equations

$$\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix} \begin{pmatrix} c_{31} \\ c_{32} \end{pmatrix} = \begin{pmatrix} a_3 \\ b_3 \end{pmatrix} \quad (6.9)$$

for the output weights c_{31} and c_{32} . Further outputs, $O_4, O_5, \text{ etc.}$, can be generated similarly, solving for output weights of each output separately. We do not need any more units to create any number of outputs.

Let us look at the 2-input case. To get the first output we know that we need 3 units. For the second output we need 3 more units(3+3). To generate the third output we need 1

more hidden unit(3+3+1). Any number of further outputs can be generated using these 7 units and solving for the output weights. The number of output weights to be solved for, is 7 in this case.

In general, for an N -input m -output case, the required number of units is given by

$$N_h(N,m) = \begin{cases} N_h(N,m-1) + \frac{(N-m+2)(N-m+3)}{2} & N-m+2 > 0 \\ N_h(N,m-1) & N-m+2 \leq 0 \end{cases} \quad (6.10)$$

and $N_h(N,1)$ is given by equation 4.8. Table 6.1 shows how the network size increases with number of inputs and number of outputs.

Table 6.1. Number of hidden units as a function of number of inputs and outputs

		Number of outputs →							
# of inputs ↓		1	2	3	4	5	6	7	8
1		1	2	2	2	2	2	2	2
2		3	6	7	7	7	7	7	7
3		6	12	14	15	16	16	16	16
4		10	20	26	29	30	30	30	30
5		15	30	40	46	49	50	50	50

Note that the number of units becomes a constant when the number of units are equal to the total number of square and cube terms

$$L_3 - N - 1 \quad (6.11)$$

where L_3 is the width of the 3-rd degree polynomial(2.2). This is obvious since the matrix of

the system of equations for the output weights becomes square only under that condition. This system of equations have to be well conditioned and consistent. If not, if they are linearly dependent, then the additional output can be generated from the units used for previous output itself by proper scaling.

Also, in the case of a 2-input case, as a particular example, it is possible to generate the third output with just 6 units instead of 7 as illustrated previously. Similar exceptions may be available in other cases too. But such ideas would make the analysis less general though. Once we get the polynomial network then it can be replaced with a sigmoidal net as illustrated in sections 5.2 and 6.1

6.3. Design procedure for large networks

Let us summarize the design procedure for constructing an N -input m -output network.

1. Design a 3-rd degree N -ary polynomial using conjugate gradient algorithm.
2. Get a polynomial perceptron using 3D_NET.
3. Replace each polynomial hidden unit in the network with sigmoidal unit.

If the hidden unit has one input follow procedure given in section 5.2.

If the hidden unit has more than one input refer to section 6.1.

4. Solve for all the output weights together using conjugate gradient algorithm after replacing all the units with sigmoid, to reduce the unit by unit mapping error.
5. Design polynomials for each output and get the sigmoidal network using the above procedure. Refer section 6.2 for networks with multiple outputs.

In terms of network efficiency, in the case of the mapping method for a function with

a square and cube term we required a sigmoidal unit with two parameters - the hidden unit threshold and the output weight of the hidden unit. That would make the network maximum efficient. Essentially the input weight is a constant and hence not counted for the efficiency calculation. If we count that as a parameter, note that we need one additional parameter for realizing the multinomial in every hidden unit. The resulting efficiency of the complete network is given by

$$\frac{N + 5}{N + 8} \quad (6.12)$$

and the efficiency increases with the number of inputs, N . The number of outputs, whether one or more than one, does not affect the efficiency of the network.

CHAPTER 7

EXAMPLES

Let us look at some examples to demonstrate the performance of the proposed methods compared to the existing techniques.

7.1. Realizing a 3-rd degree polynomial

As a first example, an arbitrary third degree univariate polynomial ($x^3 - x^2 - 9x + 9$) is realized with a network of single hidden unit (fig. 5.1) using the proposed method discussed

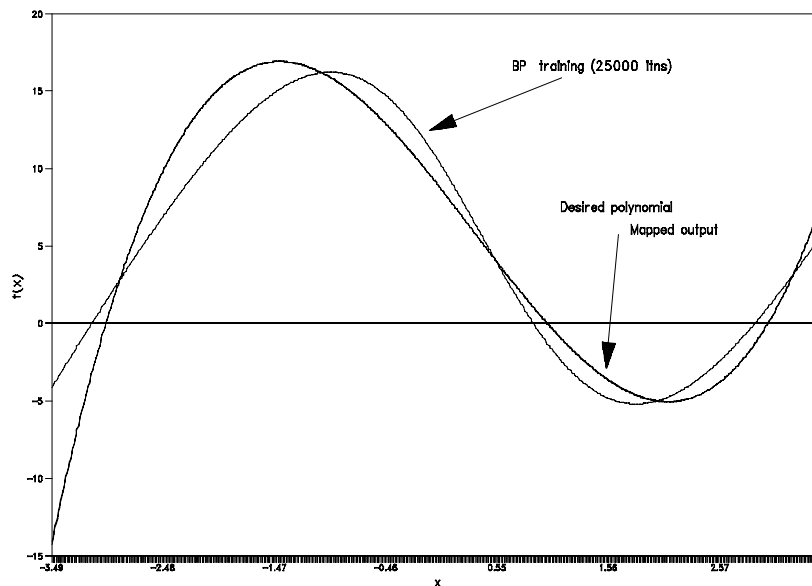


Figure 7.1. 3-rd degree polynomial realized with 1 hidden unit

in section 5.2. The mapping of the polynomial on to the network is shown in fig. 7.1. The

mapped network output is indistinguishable from the desired polynomial. The output of a network with same structure trained for 25,000 iterations using back propagation is also given for comparison purposes. The variance of the mapping error obtained by the proposed method was .005275, compared to the error variance of .28297 obtained by the back propagation training after 25,000 iterations.

7.2. Training algorithms and pattern storage

As a second example an arbitrary network is trained with four different training algorithms to look at the pattern storage obtained by them. Fig. 6.4 gives a comparison of the increase in error with the number of training patterns for different training algorithms, viz., back propagation(*BP*), output weight optimization(*OWO*), Conjugate gradient(*CG*)[10] and the proposed mapping method. The mean square error of a network increases with the number of training patterns used. The training data had 8 inputs and 1 output. Random numbers distributed between -1 and 1 were used as inputs and output. The results are given for an arbitrary three layer network with a structure 8-36-1. An 8-variable 3-rd degree equivalent polynomial is used.

Refer to table 4.2 for the bounds on an 8-input case. The network is trained for 165 iterations using the three training algorithms to be consistent with the number of iterations used in the polynomial case which uses conjugate gradient algorithm to solve for the 165 parameters or weights. The plot of *mse* of the proposed direct mapping method remains arbitrarily close to zero up to 160 patterns reaching the upper bound of pattern storage(3.3). With back propagation the error curve breaks off at about 20 patterns and it may be

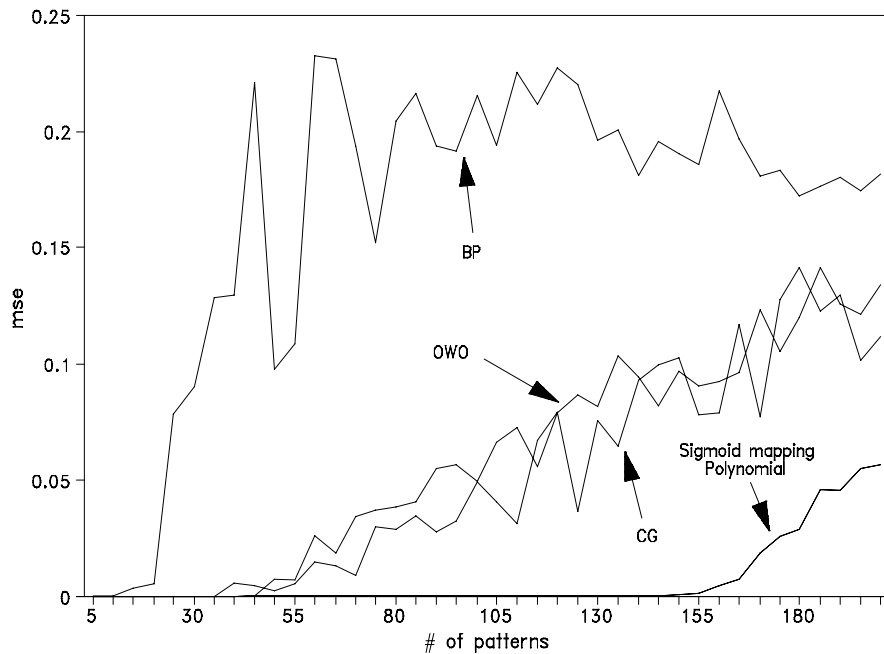


Figure 7.2. Training error vs. number of patterns

improved to attain the lower bound (*lemma 2*) with larger number of iterations. The output weight optimization and conjugate gradient are better than the back propagation as they have attained the lower bound of pattern storage. However even these algorithms lag far behind the Gabor polynomial and the proposed method. In fact, these fully connected networks have much larger number of weights, 359 in this case, than the upper bound when compared to the sparsely connected networks. These results indicate that the existing gradient training algorithms are not the final answer and certainly, better training algorithms could result in superior performance of the same network.

7.3. Network initialization

One useful application of the technique developed in this thesis is to design a good initial network for the conventional multi-layer perceptron and train the network further using back propagation or other training algorithms. It can be seen that the proposed sparsely connected is equivalent to a fully connected network with the missing connections replaced by zero weights. As an example we trained a network for a real world data with this type of initialization.

The data used was supplied by Professor, Alireza Khotanzad of Southern Methodist University, Dallas. The problem was short term hourly load forecasting in electric power system operation and planning. Load forecasting is essentially a multi-dimensional signal processing problem. Forecasting is done by extrapolating based on past load behavior along with other influencing factors. The factors which affect the load are weather parameters like temperature, relative humidity, wind speed, dew point, etc., and other factors like the particular day, time etc. The inter-relationships between the load and the factors influencing it are highly complex and non-linear. Recently, an approach based on mapping neural networks has been shown to perform exceedingly well compared to other techniques based on parametric models[62]. In this approach several individual MLP networks are trained for different hours, different days and different weeks. There were 7 weekly modules, 7 daily modules and 24 hourly modules. The training of these 38 networks everyday to adapt to the new data posed some severe problems as follows.

(1) The initial network training is very slow, and must be performed on a Decstation 5000/240 at SMU. For example, the hourly neural networks require on the order of 16,000

training iterations for satisfactory error performance. The training of all of the modules for a utility takes weeks.

(2) The required number of network hidden units differs considerably for the various utilities. Presently it is found through brute force experimentation, which is very time consuming.

To show the utility value of the theory developed in this thesis we trained an hourly module data obtained from SMU. The data had 9 inputs and one output. A network of structure 9-45-1 (N-inputs, 1-output case discussed in sections 6.1 and 6.3) was constructed. The mapping mean square errors(MSE) are given in the table 6.2. along with the training results of the same network trained with random initial weights.

Table 7.1. Mapping MSE of power load prediction data.

	50 iterations	100 iterations
Mapping	.03032	.03000
BP	.29120	.14570
OWO	.04046	.04017

The training algorithm, based upon our recent work [63] on pattern memorization and as explained earlier in this section is much faster than OWO-BP and conjugate gradient training schemes and results in a much lower error value. OWO has been able to reach an error in 600 iterations that BP reaches in 2,500 iterations. The initialization method using the same training time as 56 BP iterations, attained the performance reached by BP in 10,000 iterations. The pattern storage results also gives a fairly good idea about the number of hidden

units required for a given data set. This demonstrates the power and significant advantage of the proposed technique and its potential to develop in to a fast training algorithm for large training sets.

CHAPTER 8

CONCLUSIONS

The main issue of this thesis is the concept of pattern storage, and the approximation accuracy of MLP for a given number of weights or parameters in the network.

The bounds on the pattern storage capabilities have been established and the proposed networks are shown to be efficient in terms of pattern storage, and the number of parameters associated with them. It has been shown that an arbitrary second or third degree polynomial function can be directly mapped on to a MLP structure efficiently so as to maximize the pattern storage capability. Finally it has been shown that as the degree of mapping increases the MLP becomes less efficient than a equivalent Gabor polynomial in terms of pattern storage.

The pattern storage results also indicate that the popular training algorithm - back propagation results only in the lower bound of pattern storage. Though the output weight optimization and conjugate gradient techniques perform significantly better than BP, they also fall short of the higher bound. It is obvious that the proposed sparse network is more efficient than the normal MLP trained using back-propagation.

Efficient sigmoid networks can be found through other means resulting in higher pattern storage. The mapping method shows a possibility and proves the existence of other

training algorithms which would make the networks more efficient. The analysis can be extended to establish the efficiencies and pattern storage capabilities of higher degree networks.

There are several unsolved problems which would be interesting sequels to this work.

1. Which patterns does the network memorize or store? Is it possible to control the patterns memorized by the network while generalizing on other patterns?
2. How to eliminate the step of designing a multi-variate polynomial in designing the networks?
3. The number of hidden units increases as the square of the number of inputs. How to reduce the number of hidden units? One possible answer would be to map a higher degree polynomial directly on to a sigmoid. How to do such mapping?
4. How to make the multiple output case more efficient?
5. How to extend this idea to result in a fast training algorithm?
6. How to apply this technique to networks with more than one hidden layer and what are its implications?
6. How does a trained MLP interpolate between data compared to the multi-variate polynomial?

APPENDIX A

ALGORITHMS

A.1. Quadratic network mapping algorithm

Procedure QUAD_NET

```

call ANALYZE;

    if (zero_count = 1) call SWAP;

    elseif (zero_count > 1) call REPLACE;

     $c_1 = k_{11}$ ;

    for  $i = 2$  to  $N_{inp}$ 

         $w_{1i} = k_{1i}/2$ ; // Get the input weights of
                        // first unit

    endloop

    for  $i = 2$  to  $N_{inp}$ 

        for  $j = 1$  to  $i-1$ 

             $c_i = k_{ii} - c_j w_{ji}^2$ ; // get the output weights

        endloop

        if ( $i < N_{inp}$ ) then

            for  $k = i+1$  to  $N_{inp}$  // solve for input weights

                
$$S = \sum_{j=1}^{k-2} w_{ji} w_{jk} c_j ;$$


                 $w_{ik} = (k_{ik}/2 - S)/c_i ;$ 

            endloop

        endif

    endloop

end_Procedure; {QUAD_NET}

```

Procedure ANALYZE

```

for  $i = 1$  to  $N_{inp}$  // Count the number
    if ( $k_{ii} = 0$ ) then  $zero\_count = zero\_count + 1$ ; // of coefficients of
endloop // the  $x_i^2$  terms
end_Procedure; {ANALYZE}

```

Procedure SWAP

```

for  $i = 2$  to  $N_{inp} - 1$  // Switch inputs  $x_i$ 
    swap  $k_{ii}$  and  $k_{iN_{inp}}$ ; // (with zero
endloop // coefficient) and
end_Procedure; {SWAP} //  $x_{N_{inp}}$ 

```

Procedure REPLACE

```

for  $i = 1$  to  $N_{inp} - 1$  // Replace zero
    if ( $k_{ii} = 0$ ) then  $k_{ii} = \epsilon$ ; // coefficients with
endloop // a small positive
end_Procedure; {REPLACE} // value

```

A.2. Cubic network mapping algorithm

Procedure CUBE_NET

```

initialize all  $w$ 's and  $c$ 's with zeros;

call ANALYZE_3( $isw$ ,  $zero\_count$ ); // Count the zero

    if ( $isw > 0$ ) then call SWITCH; // coefficients

    elseif ( $zero\_count > 0$ ) then call REPLACE_3;

 $N_{units} = N_{inp}(N_{inp}+1)/2$ ;

if ( $N_{inp} > 1$ ) then // Solve for input

    for  $i = 1$  to  $N_{units} - N_{inp}$  // weights

         $k = N_{inp} - \text{round}(\frac{\sqrt{1 + 8i - 1}}{2}) + 1$  ; // k - # of
        // inputs to
        // the i-th unit

         $j = \frac{(i - N_{inp} + k - 1)(N_{inp} - k)}{2}$  ; // j - first
        // input of the
        // i-th unit

        call SOLUTION_1 ( $i, j, k$ );

        if ( $k > 2$ ) then call SOLUTION_2 ( $i, j, k$ );

         $c_i = 1$ ;

    endloop

endif

call c_SOLUTION;

end_Procedure; {CUBE_NET}

```

Procedure SOLUTION_1 (i,j,k)

$$S_1 = k_{j j+k-1} - \sum_{m=1}^{i-1} w_{mj} w_{mj} w_{m j+k-1} ;$$

$$S_2 = k_{j^{j+k-1} j+k-1} - \sum_{m=1}^{i-1} w_{mj} w_{m j+k-1} w_{m j+k-1} ;$$

$$w_{ij} = \left| \frac{S_1^2}{3S_2} \right|^{1/3} ;$$

$$w_{ik} = \frac{S_1}{3w_{ij}^2} ;$$

// Compute and solve
// equations of the form
// $x^2y^3 = k_1$
// $x^3y^2 = k_2$

if ($S_2 < 0$) **then** $w_{ij} = -w_{ij}$;

end_Procedure; {SOLUTION_1}

Procedure SOLUTION_2 (i,j,k)

for $m = j+1$ **to** $j+k-2$

$$S_1 = k_{j m j+k-1} - 6 \sum_{n=1}^i w_{nj} w_{nm} w_{n j+k-1} ;$$

$$w_{im} = S_1 / 6w_{ij}w_{i j+k-1} ;$$

// Obtain other input
// weights of the i-th
// unit

endloop

end_Procedure; {SOLUTION_2}

Procedure c_SOLUTION

for $i = 1$ **to** N_{inp} // Solve for output
// weights

$$c_{N_{units}-N_{inp}+i} = k_{iii} - \sum_{j=1}^{N_{inp}} w_{ji}^3 ;$$

$$w_{N_{units}-N_{inp}+i,i} = 1 ;$$

endloop

end_Procedure; {c_SOLUTION}

Procedure ANALYZE_3

for $i = 1$ **to** N_{inp} // count the zero

if ($k_{1iN_{inp}} = 0$) **then** $zero_count = zero_count + 1$; // coefficients,

endloop

if ($zero_count > 0$) **then** // store and set

for $i = 2$ **to** $N_{inp} - 1$ // the flags

for $j = 1$ **to** N_{inp}

if ($k_{ijN_{inp}} \neq 0$) **then** $isw = i$;

endloop

endloop

endif

end_Procedure; {ANALYZE_3}

Procedure SWITCH

```

for  $i = 1$  to  $N_{inp}$  // Switch inputs  $x_1$ 
    for  $j = i$  to  $N_{inp}$  // and  $x_{isw}$  to avoid
        swap  $k_{1ij}$  and  $k_{isw ij}$  // zero coefficients
    endloop
endloop
end_Procedure; {SWITCH}

```

Procedure REPLACE_3

```

for  $i = 1$  to  $N_{inp}$  // Replace zero
    if (  $k_{1i N_{inp}} = 0$  ) then  $k_{1i N_{inp}} = \varepsilon$  ; // coefficients with
    endloop // a small +ve
end_Subroutine; {REPLACE_3} // constant

```

A.3. Third degree network mapping algorithm

Procedure 3d_NET

```

initialize all  $w$ 's,  $a$ 's and  $b$ 's with zeros;

call ANALYZE_3( $isw$ ,  $zero\_count$ ); // Count the zero

    if ( $isw > 0$ ) then call SWITCH; // coefficients

    elseif ( $zero\_count > 0$ ) then call REPLACE_3;

 $N_{units} = N_{inp}(N_{inp}+1)/2$ ;

if ( $N_{inp} > 1$ ) then // Solve for input

    for  $i = 1$  to  $N_{units} - N_{inp}$  // weights

         $k = N_{inp} - \text{round}(\frac{\sqrt{1 + 8i} - 1}{2}) + 1$  ; // k - # of
        // inputs to
        // the i-th unit

         $j = \frac{(i - N_{inp} + k - 1)(N_{inp} - k)}{2}$  ; // j - first
        // input of the
        // i-th unit

        call SOLUTION_1 ( $i, j, k$ );

        if ( $k > 2$ ) then call SOLUTION_2 ( $i, j, k$ );

         $b_i = 1$ ;

    endloop

endif

call b_SOLUTION;

call a_SOLUTION;

end_Procedure; {3d_NET}

```

Procedure b_SOLUTION

for $i = 1$ **to** N_{inp}

$$l = N_{units} + N_{inp} + i;$$

// Solve for b
// coefficients

$$b_l = k_{iii} - \sum_{j=1}^{N_{inp}} w_{ji}^3 ;$$

$$w(l,i) = 1;$$

endloop

end_Procedure; {b_SOLUTION}

Procedure a_SOLUTION

```

l = 1; // Solve for a

for i = 1 to  $N_{inp} - 1$  // coefficients
    for j = 1 to i
         $k = N_{inp} - i + j$  // l - unit #

        
$$S_1 = \frac{k2_{jk}}{2} - \sum_{n=1}^{l-1} w_{nj} w_{nk} a_n ;$$


         $a_l = S_1 / w_{lj} w_{lk} ;$ 

        l = l + 1 ;

    endloop

endloop

for i = 1 to  $N_{inp}$ 

        
$$a_l = k2_{ii} - \sum_{n=1}^{l-1} w_{ni}^2 a_n ;$$


        l = l + 1;

    endloop

end_Procedure; {a_SOLUTION}

```

REFERENCES

- [1] V. Volterra, "Sopra le funzioni che dipendono da altre funzioni," *Rend. Regia Accademia dei Lince*, 20 Sem., pp. 97-105, 141-146, 153-158, 1887.
- [2] M. Schetzen, "Nonlinear system modeling based on the Wiener theory," *Proc. IEEE*, Vol. 69, No. 12 pp. 1457-1573, Dec. 1981.
- [3] M. Morhac, "A fast algorithm of nonlinear Volterra filtering", *IEEE Trans. on Acoustics, Speech, and Signal Proc.*, vol. 39, no. 10, October 1991, pp. 2353-2356.
- [4] E. Biglieri, "Theory of Volterra processors and some applications", *Proc. ICASSP-82*, Paris, pp. 294-297, 1982.
- [5] D. Gabor et al., "A universal nonlinear filter, predictor and simulator which optimizes itself by a learning process", *Proc. Inst. Elec. Eng.*, vol. 108B, pp. 422-438, 1961.
- [6] D.O. Hebb, *Organization of behavior*, 1948.
- [7] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," *Ph.D. dissertation*, Committee on Appl. Math., Harvard Univ., Cambridge, MA, Nov. 1974.
- [8] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel Distributed Processing*, Vol. I, Cambridge, Massachusetts: The MIT Press, 1986.

- [9] Hornick, K., M. Stinchcombe, H. White, "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, 2, 1989a, pp 359-366.
- [10] Stinchcombe, M, and White, H, "Universal Approximation Using Feedforward Networks with Non-Sigmoid Hidden Layer Activation Functions", *IJCNN 89*, vol.1, pp. 613-617, 1989.
- [11] G. Cybenko, "Approximations by Superpositions of a Sigmoidal Function", *Math. Contrl., Signals, Syst.*, Vol. 2, pp. 303-314, 1989.
- [12] Lapedes, A., and Farber, R., "Nonlinear Signal Processing using Neural Networks: Prediction and Signal Modeling", *Technical Report LA-UR-87-2662*, Los Alamos National Laboratories, Los Alamos, New Mexico, 1987.
- [13] Lippman R, "Introduction to Computing with Neural Networks", *IEEE ASSP magazine*, 1988.
- [14] D. S. Levine, *Introduction to Neural and Cognitive Modeling*, Hillsdale, NJ: Lawrence Erlbaum Assoc., 1991.
- [15] R. Hecht-Nielsen, "Theory of backpropagation neural network", in *Proceedings of IJCNN'89*, Vol. I, pp. 593-605.
- [16] T. Kohonen, "An introduction to neural computing", *Neural Networks*, Vol. 1, 1988, pp.3-16.
- [17] T. Kohonen, G. Barna, and R. Chrisley, "Statistical pattern recognition with neural networks: benchmarking studies", *IEEE International Conference on Neural Networks*, San Diego, Ca., Vol. 1, pp. 61-68, 1989.

- [18] G.A. Carpenter and S. Grossberg, "Adaptive Resonance Theory: Stable Self-Organization of Neural Recognition Codes in Response to Arbitrary Lists of Input Patterns", *Cognitive Science Society: 10th Annual Conference*, pp.45-62, 1988.
- [19] Hecht-Nielson, R., "Counterpropagation networks", *Applied Optics*, Dec.1987.
- [20] Powell, M.J.D., "Radial Basis Functions for Multivariate Interpolation", *Algorithms for Approximation*, ed. Mason and Cox, Oxford, England, pp.143-168, 1987.
- [21] Poggio, T., and Girosi, F., "Networks for Approximation and Learning", *IEEE Pae*, Sep.1990, pp.1481-1497.
- [22] Kosko, B., "Bidirectional Associative Memories", *IEEE tran. Systems, Man and Cyber.*, SMC-17, 1987
- [23] Hinton, G.E., Sejnowski, T.J., and Ackeley, D.H., "Boltzman machines: Constraint Satisfaction Networks that Learn", *Technical Report CMU-CS-84-119*, Department of Computer Science, Carnegie-Mellon University, 1984.
- [24] P.D. Wassermann, *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, New York, 1989.
- [25] Miller, W.T., Sutton, R.S., and Werbos, P.J., editors, *Neural Networks for Control*, MIT Press, Cambridge, MA, 1990.
- [26] Werbos, P., "Neuro Control and Related Techniques", *Handbook of Neural Computing Applications*, ed. A. Maren, Academic Press, New York, 1990
- [27] Brooks, R., "A Robot that Walks", *Neural Computation*, vol.1, pp.253-262, 1989.
- [28] Kung, S.Y., and Hwang, J.N., "Neural Network Architectures for Robotic Applications", *IEEE Trans. on Robotics and Automation*, vol.5, pp.641-657, 1989.

- [29] Narendra, K.S., and Parthasarathy, K., "Identification and Control of Dynamical Systems Using Neural Networks", *IEEE tran. Neural Networks*, March 1990.
- [30] Paris, B.G., Orsak et al., "Neural Net Receivers in Multiple Access Communications", *Advances in Neural Information Processing Systems I*, ed. D.S. Touretzky, Morgan Kaufman, pp.272-280, 1989.
- [31] G.G. Lorentz, "The 13th problem of Hilbert", *Proc. Symposia in Pure Mathematics*, Vol.28, pp.419-430, 1976.
- [32] Kolmogorov, Andrei Nikolaevich, "On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition", *Dokl. Nauk USSR*, 114, 953-956, 1957.
- [33] R. Hecht-Nielsen, "Kolmogorov's Mapping Neural Network Existence Theorem", *Proc. IJCNN*, vol.III, 1987, pp 11-14.
- [34] Irie, B., and Miyake, S., "Capabilities of Three-Layered Perceptrons", *IJCNN*, vol.1, pp.641-648, 1988.
- [35] S-C Huang and Y-F Huang, " Bounds on the Number of Hidden Neurons in Multilayer Perceptrons," *IEEE Transaction on Neural Networks*. Vol.2. No.1. pp. 4-55. Jan. 1991.
- [36] M.A. Sartori and P.J. Antsaklis, "A simple method to derive bounds on the size and to train multilayer neural networks," *IEEE Transactions on Neural Networks*, vol. 2, No.4, July 1991, pp. 467-471.
- [37] M.S. Chen and M.T. Manry, "Power Series Analysis of backpropagation neural networks", *Proc. IJCNN*, vol.I, 1991, pp 295-300.

- [38] M.S. Chen and M.T. Manry, "Backpropagation representation theorem using power series", *Proc. IJCNN*, vol.I, 1990, pp 643-648.
- [39] M.S. Chen and M.T. Manry, "Conventional Modeling of the Multilayer Perceptron using Polynomial basis Functions", *IEEE Trans. on Neural Networks*, vol.4, No.1, **d** 1993, pp 164-166.
- [40] Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, 65, pp.386-408, 1958.
- [41] Minsky, M., and Papert, S., *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, 1988.
- [42] Mairhuber, J.C., "On Haar's theorem concerning Chebyshev approximation problems having unique solutions", *Proc. Amer. Math. Soc.*, 7, pp.609-615, 1958.
- [43] E.W. Cheney, "Four Lectures on Multivariate Approximation", *Approximation Theory and Spline Functions*, ed. S.P. Singh, J.W.H. Barney, and B. Watson, D. R e i d e l Publishing Company, Holland, 1984. pp. 65-88.
- [44] R.R. Goldberg, *Methods of Real Analysis*, John Wiley and Sons, New York, 1976.
- [45] Llavona J.G., "Approximations of Differentiable Functions", *Studies in Analysis, Advances in Mathematical Supplemental Studies*, ed. G.C. Rota, vol.4, Academic Press, NY, 1979.
- [46] Y-H Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley Publishing Company, Inc., New York, 1989.

- [47] C.K.Chui, M.J. Lai, "Vander Monde Determinants and Lagrange Interpolation in \mathbf{R}^n ", *Nonlinear and convex analysis*, ed. B.C. Lin, S.Simons, Marcel Dekker Inc., 1988. pp. 23-35.
- [48] M.T. Manry, X. Guan, S.J. Apollo, L.S. Allen, W.D. Lyle, and W. Gong, "Output weight optimization for the multi-layer perceptron," *Conference Record of the Twenty-Sixth Annual Asilomar Conference on Signals, Systems, and Computers*, Oct. 1992, vol 1, pp. 502-506.
- [49] E. Barnard, "Optimization for Training Neural Nets", *IEEE transactions on Neural Networks*, Vol. 3, No.2, Mar 1992, pp 232-240.
- [50] P.J. Werbos, "Backpropagation through time: what it does and how to do it", *Proc. of the IEEE*, October 1990, pp. 1550-1560.
- [51] T. M. Jelonek and James P. Reilly, "Maximum likelihood estimation for direction of arrival using a nonlinear optimizing neural network", *Intrnl. Joint Conf. on Neural Networks*, vol. I, pp 253-258, 1990.
- [52] Mu-Song Chen, "Analysis and design of Multi-layer perceptron using Polynomial Basis Function", phd. dissertation, The University of Texas at Arlington, Dec. 1991
- [53] A. Clebsch, "Über Curven vierter Ordnung, *J. Reine Angew Math.* 59, 1861 pp 125-145.
- [54] J.J. Sylvester, "On a remarkable discovery in the theory of canonical forms and of hyperdeterminants", *Phil. Mag.* 2, 1851, pp 125-145. (Reprinted in "*Collected papers*", Cambridge University press, vol.1, 41, 1904, pp 265-283.)

- [55] J.J. Sylvester, Sur une extension d'un théorème de Clebsch relatif aux courbes du quatrième degré, *C.R. Acad. Sci.* 102, 1886, pp 1532-1534. (Reprinted in "*Collected papers*", Cambridge University press, vol.4, 47, 1912. pp 527-528.)
- [56] J. Wray and G.G.R. Green, "The Practical Use of Artificial Networks: An Investigation Using Taylor Series Expansions of The Network Equations," *Proc. of IJCNN'91*, Seattle WA., pp. II A-995.
- [57] G.W. Davis and M.L.Gasperi, "ANN modelling of Volterra Systems", *Proc. IJCNN*, vol.II, 1987, pp 727-734.
- [58] K. Rohani, M.S. Chen and M.T. Manry, "Neural Subnet Design by Direct Polynomial Mapping," *IEEE Transactions on Neural Networks*, Vol. 3, no. 6, pp. 1024-1026, November 1992.
- [59] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients", *Computer J.*, Vol. 7, pp. 149-154, 1964.
- [60] R. Fletcher, "Conjugate direction methods," in *Numerical Methods for Unconstrained Optimization*, W. Murray, Ed. London and New York: Academic Press, pp 73-86, 1972.
- [61] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press, New York, 1981.
- [62] A. Khotanzad, R-C Hwang, and D. Maratukulam, "Hourly Load Forecasting by Neural Networks," *IEEE PES Winter Meeting*, Columbus Ohio, February 1993.

- [63] A. Gopalakrishnan, X. Jiang, M-S Chen, and M.T. Manry, "Constructive Proof of Efficient Pattern storage in the Multilayer Perceptron," *Conference Record of the Twenty-Seventh Annual Asilomar Conference on Signals, Systems, and Computers*, Nov. 1993.