PERGAMON

Neural Networks 14 (2001) 495–504

Neural Networks

www.elsevier.com/locate/neunet

Contributed article

A model with an intrinsic property of learning higher order correlations

Marifi Güler*

Department of Computer Engineering, Eastern Mediterranean University, Famagusta, Mersin -10, Turkey

Received 1 June 2000; accepted 12 January 2001

Abstract

A neural network model that can learn higher order correlations within the input data without suffering from the combinatorial explosion problem is introduced. The number of parameters scales as $\tilde{M} \times N$, where $\tilde{M}$ is the number such that no higher order network with less than $\tilde{M}$ higher order terms can implement the same input data set and N is the dimensionality of the input vectors. In order to have better generalization, the model was designed to realize a supervised learning such that after learning, output for any input vector is the same as the output of a higher order network that implements the same input data set using $\tilde{M}$ number of higher order terms. Unlike the case in product units, the local minima problem does not pose itself as a severe problem in the model. Simulation results for some problems are presented and the results are compared with the results of a multilayer feedforward network. It is observed that the model can generalize better than the multilayer feedforward network.

Keywords: Higher order network; Higher order term; Monomial; Relevant set; Minimal set; Product unit; Local minima; Generalization

1. Introduction

Studies by various authors including Durbin and Rumelhart (1989); Giles and Maxwell (1987); Mel (1990); Personnaz, Guyon and Dreyfus (1987); Redding, Kowalczyk and Downs (1993) and Taylor and Coombes (1993) suggest that higher order networks can be more powerful and are biologically more plausible with respect to the more traditional multilayer networks, and higher order associative memories have a memory storage capacity significantly better than the capacity of the Hopfield model (Abbot & Arian, 1987; Baldi & Venkatesh, 1987; Burshtein, 1998; Gardner, 1987; Lee et al., 1986; Peretto & Niez, 1986; Psaltis & Park, 1986). These architectures make explicit use of nonlinear interactions between input variables in the form of higher order units. Higher order units are capable of implementing functions $\phi : \{-1, 1\}^N \rightarrow \mathscr{R}$ by means of the finite expansion

$$\phi(\mathbf{x}) = \sum_{\alpha} s_\alpha m^\alpha(\mathbf{x}); \qquad \mathbf{x} \in \{-1, 1\}^N, \qquad s_\alpha \in \mathscr{R} \quad (1)$$

where

$$m^\alpha(\mathbf{x}) = \prod_{j=1}^{N} x_j^{\alpha_j}; \qquad \alpha = \alpha_1 \ldots \alpha_N \in \{0, 1\}^N \quad (2)$$

are called monomials. The expansion runs over all realizations of binary strings α and, therefore, contains 2^N higher order terms (weighted monomials). Usually, but not necessarily, the value in Eq. (1) is passed through the *signum* function and, consequently, the output of a higher order unit is $\{-1, 1\}$. Then, a higher order network consisting of P higher order units can implement any function $\phi : \{-1, 1\}^N \rightarrow \{-1, 1\}^P$. Due to the combinatorial explosion of the number of higher order terms in the dimensionality of the inputs, being able to learn a set of higher order terms relevant to the implementation of a specific logical function is important. For example, detecting the monomial with $\alpha_j = 1$; $j = 1, \ldots, N$, in Eq. (2) is all needed for the implementation of the parity problem.

If it is known a priori that the problem to be implemented possesses a given set of invariances like in the translation, rotation, and scale invariant pattern recognition problems, those invariances can be encoded, thus eliminating all higher order terms which are incompatible with the invariances (Giles & Maxwell, 1987; Perantonis & Lisboa, 1992; Spirkovska & Reid, 1993). In general, however, higher order networks should be capable of learning a set of higher order terms required for the implementation of a specific logical function. This is due to the fact that, even though the number of higher order terms increases exponentially with the size of inputs, usually only a small number of those terms are needed to implement a certain logical function.

* Tel.: +90-392-630-1120; fax: +90-392-365-0711.
E-mail address: marifi.guler@emu.edu.tr (M. Güler).

0893-6080/01/$ - see front matter
PII: S0893-6080(01)00033-8

Nomenclature

Σ	Summation
$\prod$	Product
$\mathscr{R}$	Set of real numbers
$\|.\|$	Absolute value
sgn(.)	The signum function
tanh(.)	Hyperbolic tangent function
Dtanh(.)	Derivative of tanh(.) with respect to its argument
$\mathbf{x}$	Input vector with the bipolar components $\{-1, 1\}$
x_i	ith component of the input vector $\mathbf{x}$
$\mathbf{p}$	A training vector
p_i	ith component of the input vector $\mathbf{p}$
$d_\mathbf{p}$	Target output for the training vector $\mathbf{p}$
P	Training set
N	Dimensionality of the input vectors
M	Number of hidden nodes in the network
$\Delta^\mathbf{p}$	Weight adjustment value for the training vector $\mathbf{p}$

Besides, the VC dimension of higher order networks has been computed as the number of higher order terms that the network employs (Young & Downs, 1993). That is to say, a higher order network that implements the training data set using the minimum number of monomials possible is expected to give a learning curve with a more rapidly rising correct classification rate than the learning curve of any other higher order network that employs a greater number of monomials. Thus, for better generalization, as well as for less storage space, the problem to be solved, in supervised learning, is how to find a minimal set of monomials without suffering from the combinatorial explosion of the number of higher order terms in the input dimensionality. Here, what is meant by a *minimal set* of monomials is that a set of monomials that, with an appropriate set of values of weights, can implement the training data set such that no set of monomials with less number of monomials than the number of monomials in a minimal set can implement the same training data set. The problem has been a major bottleneck for higher order networks. Note that having a minimal set with a small number of monomials does not solve the combinatorial explosion problem in higher order networks. This is due to the fact that, since the nonzero weights are not known in advance, the higher order network still has to cope with 2^N weight parameters.

Networks that use product units (Durbin & Rumelhart, 1989) or product terms (Güler & Şahin, 1994) are, in principle, capable of learning a set of monomials that implements the training data set without suffering from the combinatorial explosion problem. Those approaches use an implicit representation of higher order terms and expect the network to learn the monomials as well as the associated weight values. In the former, monomials are specified through a *cosine* activation function and, in the latter, each input component is coupled with an additive coefficient and then their product is taken. However, problems are encountered when using the backpropagation algorithm to train networks containing product units or product terms. In the case of product units, using a *cosine* activation function often leads to numerical problems, and nonglobal minima, whereas with sigmoids the problem is not so severe (Lapedes & Farber, 1987; Leerink, Giles, Horne & Jabri, 1995). A similar problem, due to the product form of the adjustable weights, also arises for the networks containing product terms. Another fundamental difficulty with product units is that using a unit with a *cosine* activation function makes that unit's VC dimension infinite (Brady, 1990). That is to say, product units will generalize very poorly unless the weight parameters are allowed to take only a certain finite number of possible values such as 0 and 1, but it is very difficult to force adjustable parameters to certain discrete values in a gradient descent learning algorithm without seriously suffering from the local minima problem.

Models, as in Fahner and Eckmiller (1994) and Redding et al. (1993), that make direct use of expansion (1) and, yet, avoid the combinatorial explosion of the terms have been proposed. These models are based on finding one set among a large number of relevant sets of monomials that implement the training data set. Here, what is meant by a *relevant set* of monomials is that if one element from the set is removed, the remaining set of monomials cannot implement the same training data set with any choice of the weight values. Note that the number of relevant sets will be very large when the number of patterns in the training data set is significantly less than 2^N, and only one, or few of the relevant sets will be a minimal set. For a simple example, take, in $N=2$, the training patterns $(1, 1) \rightarrow 1$, $(-1, -1) \rightarrow 1$, and $(-1, 1) \rightarrow -1$. There are two relevant sets of monomials as $\{x_1x_2\}$ and $\{1, x_1, x_2\}$ of which, only the first one is a minimal set. Therefore, these models have the shortcoming of ending up with a relevant set containing much larger number of monomials than the number of monomials in a minimal set. In addition, as the number of patterns in the training data set becomes relatively larger, the number of relevant sets decreases and, consequently, computational cost of finding a relevant set becomes extremely expensive.

In this paper, we introduce a neural network model that tries to implement the training data set by learning a minimal set and does this without suffering from the combinatorial explosion of the number of parameters in the input dimensionality. The idea is similar to the idea behind the product units that a single unit which represents a single higher order term, but learns which one to represent. However, the representation we use is entirely different than the representation of product units. Product units can be thought of as formal neurons with a *cosine* activation function. In the representation we propose, there is no such single activation function, but rather there is a superposition of some functions. The

reason behind that representation is to enable learning a minimal set without severely suffering from the local minima problem. A learning algorithm that forces finding a minimal set rather than a relevant set is designed.

In Section 2, the theory behind the representation that forms the basics of the model is introduced. In Section 3, the structure of the model is introduced. In that section, the model is also presented as a network. In Section 4, a gradient descent learning algorithm for supervised learning by the model is derived. In Section 5, simulation results are presented. Generalization of the model is also compared with the generalization of the multilayer feedforward network. Finally, in Section 6, discussion and conclusions are given.

2. Theory of the model

In this section, we introduce a theorem that forms the basics of the model.

Theorem 2.1. Let $A(\mathbf{x})$ be defined by

$$A(\mathbf{x}) = g \sum_{l=-\infty}^{+\infty} (-1)^{|l|} \mathrm{e}^{-|B(\mathbf{x})-2l|}, \tag{3}$$

$g \in \mathscr{R}, \qquad \mathbf{x} \in \{-1, 1\}^N$

and

$$B(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^{N} (\omega^i + 1) x_i - \frac{1}{2}(v + 1) \tag{4}$$

where ω^i $(i = 1, \ldots, N) \in \{-1, 1\}$, $v \in \{-1, 1\}$, and $|.|$ denotes the absolute value. Then, there is an equivalence between $A(\mathbf{x})$ and the higher order terms in expansion (1). That is, any higher order term can be implemented by $A(\mathbf{x})$ with an appropriate choice of values of parameters g, ω^i $(i = 1, \ldots, N)$, and v; and that, for a given set of values of the parameters, no more than one higher order term is implemented by $A(\mathbf{x})$.

Proof. Assume that the higher order term to be implemented is $sx_{k_1}x_{k_2}\cdots x_{k_r}$ where let r be a positive even integer. Let $\omega^i = 1$ for $i = k_1, k_2, \ldots, k_r$, and $\omega^i = -1$ otherwise. Also, let $v = -1$. Then, $B(\mathbf{x}) = x_{k_1} + x_{k_2} + \cdots + x_{k_r}$. Since r is even, there exists an integer l^* such that $B(\mathbf{x}) = 2l^*$. Then, making the substitution $j = l - l^*$ in Eq. (3) gives

$$A(\mathbf{x}) = g \sum_{j=-\infty}^{+\infty} (-1)^{|j+l^*|} \mathrm{e}^{-2|j|} \tag{5}$$

where l^* is a function of $\mathbf{x}$. Splitting the infinite sum in Eq. (5) into three parts; one with $j = 0$, one with the negative j values, and one with the positive j values; and, then, making use of the absolute values gives

$$A(\mathbf{x}) = g\,(-1)^{|l^*|}\left(1 - 2\mathrm{e}^{-2}(1 - \mathrm{e}^{-2}) \sum_{n=0}^{+\infty} \mathrm{e}^{-4n}\right) \tag{6}$$

Then, substitution of the value of the infinite sum in Eq. (6) results in

$$A(\mathbf{x}) = g(-1)^{|l^*|}(1 - 2e^{-2}(1 - e^{-2})/(1 - e^{-4})) \tag{7}$$

The higher order term $sx_{k_1}x_{k_2}\cdots x_{k_r}$ has the value $-s$ or s depending on the given $\mathbf{x}$. After setting g appropriately, $A(\mathbf{x})$ will also have that value. Consider the transformation $x_{k_t} \rightarrow -x_{k_t}$ in $\mathbf{x}$ for a choice of t. The transformation results in $B(\mathbf{x}) = 2(l^* - 1)$ or $B(\mathbf{x}) = 2(l^* + 1)$, that is $l^* \rightarrow l^* - 1$ or $l^* \rightarrow l^* + 1$, if $1 \le t \le r$; and results in $B(\mathbf{x}) = 2l^*$, with no change in l^*, if $t \ge r$. Consequently, $A(\mathbf{x})$ transforms as $A(\mathbf{x}) \rightarrow -A(\mathbf{x})$ if $1 \le t \le r$; and $A(\mathbf{x}) \rightarrow A(\mathbf{x})$ if $t \ge r$. The higher order term $sx_{k_1}x_{k_2}\cdots x_{k_r}$ also transforms in the same way. Hence, $A(\mathbf{x})$, with the given set of parameter values, and the higher order term $sx_{k_1}x_{k_2}\cdots x_{k_r}$, with an even r, are equivalent. In the case of r is being odd, setting $v = 1$, instead of $v = -1$, and following the same steps also results in Eq. (7). Implementation of the constant higher order term s is straightforward: simply by setting $\omega^i = -1$, $i = 1, \ldots, N$, and $v = -1$. Thus, any higher order term can be implemented by $A(\mathbf{x})$ with an appropriate choice of values of the parameters g, $\omega^i (i = 1, \ldots, N)$, and v. This completes the first part of the proof. In the case of having an even number of $\omega^i = 1$s and $v = 1$ in $B(\mathbf{x})$, after some algebra, we obtain $A(\mathbf{x}) = 0$. Similarly, also in the case of having an odd number of $\omega^i = -1$s and $v = -1$ in $B(\mathbf{x})$, one obtains $A(\mathbf{x}) = 0$. This completes the proof.

It is important that, for a given set of values of the parameters, $A(\mathbf{x})$ cannot implement more than one higher order term. We will make use of this property of $A(\mathbf{x})$ in order to force the model to find a minimal set rather than a relevant set.

3. Structure of the model

We introduce the model for the implementation of the functions $\phi: \{-1, 1\}^N \rightarrow \{-1, 1\}$. Extension to the case of $\phi: \{-1, 1\}^N \rightarrow \{-1, 1\}^P$ is straightforward. The structure is based on Eqs. (3) and (4). Since $A(\mathbf{x})$ can implement any, but only one, higher order term, the model incorporates some M number of those $A(\mathbf{x})$s, which we label as $A_m(\mathbf{x})$ $(m = 1, \ldots, M)$, where M is not to be smaller than the minimal number of higher order terms essential for the implementation of the training data set in consideration. Correspondingly, there are M number of $B(\mathbf{x})$s labelled as $B_m(\mathbf{x})$ $(m = 1, \ldots, M)$. Consequently, the adjustable parameters ω^i $(i = 1, \ldots, N)$ and v in Eq. (4) will also have the associated index m. These parameters, however, need to be converted into reals; so that they can be updated gradually in a learning algorithm. We

do this by making use of the tanh function:

$$B_m(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{N}\Big(\kappa\tanh(\gamma\omega_m^i) + 1\Big)x_i - \frac{1}{2}(\kappa\tanh(\gamma v_m) + 1) \tag{8}$$

where ω_m^i and v_m are now reals. κ and γ are constant parameters. κ is to be assigned a value slightly greater than 1. The reason for the inclusion of κ is that there shall be no need for the tanh(.)s to saturate to the limiting values -1 and $+1$. γ simply specifies the steepness of the tanh(.)s. $A_m(\mathbf{x})$ ($m = 1, \ldots, M$) are defined as follows:

$$A_m(\mathbf{x}) = \sum_{l=q(\mathbf{x})}^{z(\mathbf{x})}(-1)^{|l|}\tanh\Big(\gamma|\theta_m^l|\Big)e^{-|B_m(\mathbf{x})-2l|} \tag{9}$$

Unlike the sum in $A(\mathbf{x})$, in Eq. (3), the sum is now a finite sum so that $A_m(\mathbf{x})$ can be computed in finite time. The lower bound $q(\mathbf{x})$ and the upper bound $z(\mathbf{x})$ are functions of the input vector $\mathbf{x}$, set as follows:

$q(\mathbf{x})$ is the smallest integer not less than $\dfrac{-n^-(\mathbf{x}) - 5}{2}$

$z(\mathbf{x})$ is the largest integer not more than $\dfrac{n^+(\mathbf{x}) + 4}{2}$

where $n^+(\mathbf{x})$ and $n^-(\mathbf{x})$ denote the number of $+1$s and the number of -1s in the input vector $\mathbf{x}$, respectively. $q(\mathbf{x})$ and $z(\mathbf{x})$ are decided by ignoring those terms with

$$|B(\mathbf{x}) - 2l| \geq 4$$

in the infinite sum in Eq. (3). After observing that the minimum and the maximum possible values for $B_m(\mathbf{x})$, in case of $|\kappa\tanh(.)| = 1$, are $-n^-(\mathbf{x}) - 1$ and $n^+(\mathbf{x})$, respectively; the above values for $q(\mathbf{x})$ and $z(\mathbf{x})$ are easily concluded.

The adjustable parameters θ_m^l introduced in $A_m(\mathbf{x})$ in Eq. (9) are, in principle, not needed. They are, however, introduced in connection with the local minima problem. The idea is that, in a gradient descent learning algorithm, they shall initially be set to values around 0 and, as the learning takes place, let them *bubble up* gradually as they become necessary. Therefore, one expects the learning to take place without much of a problem of getting stuck in a local minimum. We have observed throughout the simulations that this is in fact the case and the presence of those adjustable parameters helps in the avoidance of the local minima problem. The indices in θ_m^l have the ranges $m = 1, \ldots, M$, and $l = l_i, \ldots, l_f$ where

l_i is the smallest integer not less than $\dfrac{-N - 5}{2}$

l_f is the largest integer not more than $\dfrac{N + 4}{2}$

Note that, l_i and l_f correspond to the minimum possible value of $q(\mathbf{x})$ and the maximum possible value of $z(\mathbf{x})$, for any $\mathbf{x}$, respectively.

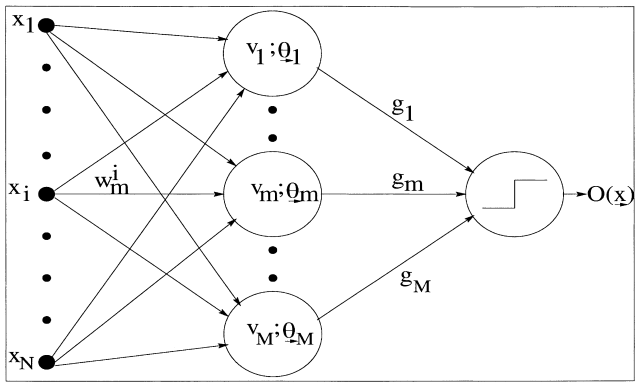

Fig. 1. Architectural graph of this network.

The output of the model, denoted by $O(\mathbf{x})$ for the input vector $\mathbf{x}$, is

$$O(\mathbf{x}) = sgn\left(\sum_{m=1}^{M} g_m A_m(\mathbf{x})\right) \tag{10}$$

where *sgn* denotes the *signum* function and g_m ($m = 1, \ldots, M$) are some adjustable parameters. Note that, even though the parameter g is a part of $A(\mathbf{x})$ in Eq. (3), g_m are not included in $A_m(\mathbf{x})$ in Eq. (9). This is just for convenience and will serve better for the rest of the paper.

The model can be visualized as a network with the architecture as given in Fig. 1. The network consists of N input nodes, M hidden nodes, and a single output node. The adjustable parameters ω_m^i and g_m are connection weights; ω_m^i is the connection weight from the ith input node to the mth hidden node, and g_m is the connection weight from the mth hidden node to the output node. Adjustable parameters v_m and the adjustable vectors θ_m are adjustable internal parameters for the mth hidden node. The mth hidden node computes $A_m(\mathbf{x})$ as its output. The output node is a formal neuron; it simply computes the weighted sum of the outputs from the hidden nodes and passes it through the *signum* function.

The complexity of the network topology is of the order $M \times N$. That is, the number of parameters in the model linearly increases with the number of higher order terms that the network is expected to implement, and, also, linearly increases with the dimensionality of the inputs. There is no problem of combinatorial explosion of terms or parameters in the input dimensionality.

No continuously oscillating functions such as the *cosine* function have been used in the model and, therefore, a severe suffering from the local minima problem is not expected (Lapedes & Farber, 1987); it is a known fact that neural networks that employ nonmonotonic transfer functions are more difficult to train than networks that use monotonic transfer functions, because the former can be expected to have more local minima.

4. Learning algorithm

In supervised learning, normally, a cost function is described and minimized through the gradient descent technique. We employ this technique here using the following cost function

$$E = E_1 + E_2 \tag{11}$$

where E_1 enables learning the training data set while E_2 performs weight elimination. E_1 is given by

$$E_1 = \frac{1}{2}\sum_{\mathbf{p}\in P}(d_{\mathbf{p}} - \tanh(\beta S_1(\mathbf{p}))^2(d_{\mathbf{p}} - S_1(\mathbf{p}))^2 \tag{12}$$

where β is a constant. P is the training set consisting of the patterns $(\mathbf{p}, d_{\mathbf{p}})$, where $\mathbf{p}$ is a training vector with the associated target output value $d_{\mathbf{p}}$. $S_1(\mathbf{p})$ is the weighted sum of the outputs from the hidden nodes as given by

$$S_1(\mathbf{p}) = \sum_{m=1}^{M} g_m A_m(\mathbf{p}) \tag{13}$$

In E_1, the use of tanh(.) is due to the fact that the output of the network is $\{+1, -1\}$ and that tanh(.) is a differentiable function that approximates the *signum* function. Even though the second quadratic term in Eq. (12) is not essential, it is, however, included just to give a chance to change to the adjustable parameters even after a possible incorrect saturation of the $\tanh(\beta S_1(\mathbf{p}))$; note that the derivative of $\tanh(x)$ is negligible when $|x|$ is large. However, this is a minor point and the second quadratic term can be ignored in principle without any problem.

The reason for having E_2 in Eq. (11) is to perform weight elimination (Weigend, Rumelhart & Huberman, 1990) on the $\mathbf{g}$ weights connecting the hidden nodes to the output node. Thus, E_2 is given by

$$E_2 = \frac{1}{2}\lambda\sum_{m=1}^{M}\frac{(g_m/g^*)^2}{1+(g_m/g^*)^2} \tag{14}$$

where λ is a regularization parameter and g^* is a weight normalization factor. E_2 forces the network to make use of the minimal number of hidden nodes. Note that we apply the weight elimination only on the $\mathbf{g}$ weights and not on the other adjustable parameters. The reason for this should be clear from Theorem 2.1. In the theorem, it was not only proven that any higher order term can be implemented by $A(\mathbf{x})$, but it was also proven that $A(\mathbf{x})$ cannot implement more than one higher order term for a given set of adjustable parameter values. Thus, the weight elimination on the $\mathbf{g}$ weights forces the network to implement the training data using the minimal number of $A_m(\mathbf{x})$s and, therefore, using the minimum number of monomials possible. That is why $A(\mathbf{x})$ was designed such that it could not implement more than one higher order term for a given set of adjustable parameter values. Note that, even though the weight elimination process is not guaranteed to find a minimal solution, it increases the chance of the result being a minimal or nearly minimal solution.

The weight adjustments that minimize the cost function E in the gradient descent technique, for the training vector $\mathbf{p}$, are given as follows:

$$\Delta^{\mathbf{p}} g_m = \eta S_2(\mathbf{p}) A_m(\mathbf{p}) - \frac{\eta\lambda}{g^*}\frac{g_m/g^*}{(1+(g_m/g^*)^2)^2} \tag{15}$$

$$\Delta^{\mathbf{p}} v_m = -C_m(\mathbf{p}) D\tanh(\gamma\theta_{m^l}) \tag{16}$$

$$\Delta^{\mathbf{p}}\omega_m^i = C_m(\mathbf{p}) D\tanh(\gamma\omega_m^i)\mathbf{p}_i \tag{17}$$

$$\Delta^{\mathbf{p}}\theta_m^l = \begin{cases} \eta\gamma S_2(\mathbf{p}) g_m (-1)^{|l|} sgn(\theta_m^l) D\tanh(\gamma\theta_{m^l}) e^{-|B_m(\mathbf{p})-2l|}, \\ 0, \\ \text{for } z(\mathbf{p}) \le l \le q(\mathbf{p}) \\ \text{otherwise} \end{cases} \tag{18}$$

where η is the learning rate. In these equations the ranges of the indices are as follows; $i = 1, \ldots, N$; $m = 1, \ldots, M$; and $l = l_i, \ldots, l_f$. $D\tanh(x)$ denotes the derivative of $\tanh(x)$ with respect to the argument x. $S_2(\mathbf{p})$ and $C_m(\mathbf{p})$ denote the expressions

$$S_2(\mathbf{p}) = (d_{\mathbf{p}} - \tanh(\beta S_1(\mathbf{p}))(d_{\mathbf{p}} - S_1\mathbf{lp}))(\beta(d_{\mathbf{p}} - S_1(\mathbf{p}))D\tanh(\beta S_1(\mathbf{p})) + (d_{\mathbf{p}} - \tanh(\beta S_1(\mathbf{p})))) \tag{19}$$

and

$$C_m(\mathbf{p}) = -\frac{1}{2}\eta\kappa\gamma S_2(\mathbf{p}) g_m \sum_{l=q(\mathbf{p})}^{z(\mathbf{p})} (-1)^{|l|}\tanh\left(\gamma|\theta_m^l|\right) sgn(B_m(\mathbf{p}) - 2l) e^{-|B_m(\mathbf{p})-2l|} \tag{20}$$

respectively.

5. Simulations

In this section, we simulate the model and test its performance on some problems. The parameters in the model are set to the following values: $\eta = 0.001$, $\lambda = 0.005$, $\kappa = 1.2$, $\gamma = 3.0$, $\beta = 3.0$, and $g^* = 1.0$. The backpropagation algorithm in on-line training mode using the weight adjustments in Eqs. (15)– (18) is applied for training the network. All the adjustable parameters are initially set to some randomly selected values within the domain $[-0.1, 0.1]$.

In order to study the effect of an increase in N, we have taken the problem described by the higher order term $x_1x_2x_3x_4x_5$, and studied N versus the minimum number of hidden nodes M necessary for the model to learn the problem. Through the observations for $N = 10, 15, 20, 25$ values, we have found that the minimum number of hidden nodes is

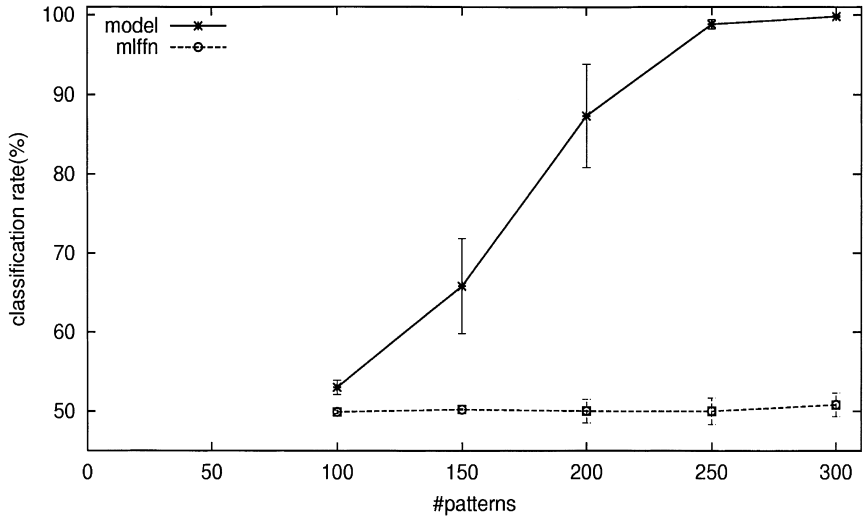

Fig. 2. Generalization for Problem 1 defined by the full parity function $\prod_{j=1}^{20} x_j$ in $N = 20$ dimensions. Performance of the model is compared with the performance of the multilayer feedforward network (mlffn).

dependent on the initial values of the adjustable parameters and the number of training patterns, but there was no clear dependence on the value of N. For example, we have observed cases that where $N = 15$, $M = 1$ hidden node was sufficient and, on the other hand, for $N = 10$, $M = 4$ hidden nodes were necessary. Throughout 24 different runs, the necessary number of hidde nodes varied between $M = 1$ and $M = 8$. The extra hidden nodes, were, however, disabled through the learning process as a consequence of the weight elimination. Even though theoretically $M = 1$ hidden node is sufficient for this problem, the need for up to eight hidden nodes should not come as a surprise; similarly, in multilayer networks, it is a well known fact that, usually, a network topology larger than the theoretical minimal is necessary for the avoidance of the local minima.

We have compared the model's performance to generalize with the performance of the standard multilayer feedforward network with a single hidden layer. The nodes in the multilayer network are formal neurons with the logistic activation function. The multilayer network is trained using the back-propagation algorithm, where, in addition to the usual quadratic cost function, a second cost that performs weight elimination is also included. The problems for which we present simulation results on generalization are in $N = 20$ dimensions. There are six problems defined as follows:

- **Problem 1**: This is the full parity problem defined by $\prod_{j=1}^{20} x_j$.
- **Problem 2**: This is a partial parity problem defined by $\prod_{j=1}^{15} x_j$. Note that the input dimension is still $N = 20$, but the input channels $x_{16}, x_{17}, x_{18}, x_{19}, x_{20}$ have no effect on the target output. The reason for testing this problem was to see the model's ability in detecting the redundant input channels.
- **Problem 3**: This is a problem described by five higher order terms as follows: $sgn(1 + \prod_{j=1}^{2} x_j + \prod_{j=1}^{3} x_j + \prod_{j=1}^{19} x_j + \prod_{j=1}^{20} x_j)$. The reason for testing this problem was to see how the model behaves when low order and high order terms are present in the underlying function simultaneously.
- **Problem 4**: This is a problem described by the higher order terms as follows: $sgn(1 + \prod_{j=1}^{2} x_j + \prod_{j=1}^{3} x_j + \prod_{j=1}^{4} x_j + \prod_{j=1}^{5} x_j) + \prod_{j=1}^{17} x_j + \prod_{j=1}^{18} x_j + \prod_{j=1}^{19} x_j + \prod_{j=1}^{20} x_j)$. The reason for testing this problem was to see the effect of an increase in the number of terms in the underlying function.
- **Problem 5**: This is the left-right-shift problem in $N = 20$ dimensions. This problem is described as follows: in a vector, the sequence of the second 10 bits is essentially a repetition of the first 10 bits, but cyclically shifted for one place either to the left or to the right. The target output is either $+1$ or -1 depending on the left or right shift. The four vectors $(1, 1, 1, 1, \ldots)$, $(-1, -1, -1, -1, \ldots)$, $(1, -1, 1, -1, \ldots)$, and $(-1, 1, -1, 1, \ldots)$ are excluded from the set of input vectors since the left shift and the right shift operations give the same result for those strings. The reason for testing this problem was to see how the model performs for a low order problem. Note that the left-right-shift problem is a second order problem, that is, it is implemented by means of the second order higher order terms.
- **Problem 6**: In this problem, the input vector set is essentially the same as the input vector set of Problem 5, but it is constructed as follows: if, in a vector, the first six bits contain an even (odd) number of -1s and the second 10 bits is obtained by the left (right) shift of the first 10 bits then the target output is $+1$, otherwise the target output is -1. The reason for testing this problem was to see the effect of the increase in the order of the problem.

Generalization results for these problems are given in Figs. 2–7. Each figure is a plot of number of training

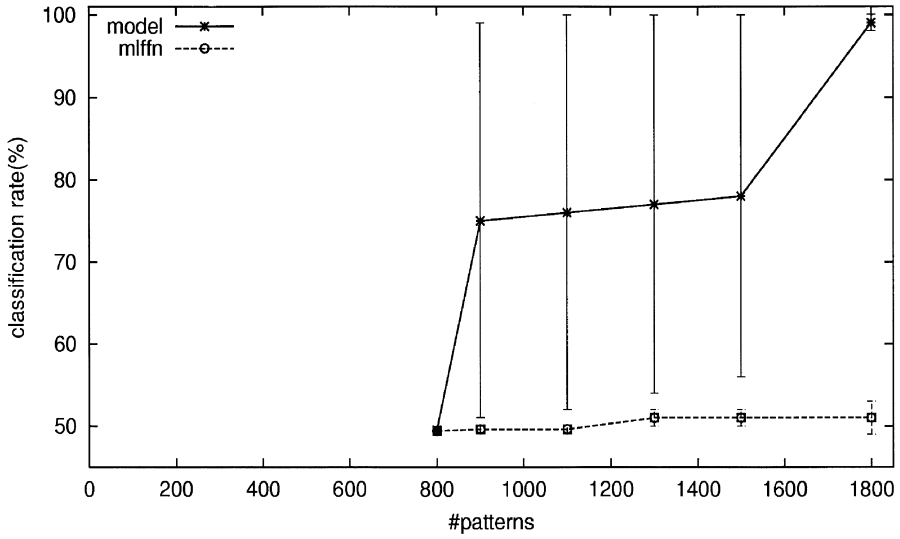

Fig. 3. Generalization for Problem 2 defined by the partial parity function $\prod_{j=1}^{15} x_j$ in $N = 20$ dimensions. Performance of the model is compared with the performance of the multilayer feedforward network (mlffn).

patterns versus correct classification rate on the test set, where *model* and *mlffn* denote the model and the multilayer feedforward network, respectively. The error bars shown for each point were obtained from four runs, where, each run corresponds to different initial values of the adjustable parameters. The vectors in the training set and the test set were selected randomly. The training set and the test set were taken as two disjoint sets; a pattern that is included in the training set was not allowed to be included also in the test set. The test set contained 10,000 patterns for Problems 1–4. For Problems 5 and 6, the number of well-defined patterns is 2040; the test set contained all those well-defined patterns that are not included in the training set. The number of hidden nodes in the model was set to $M = 30$. For the multilayer network, on the other hand, a large number of hidden nodes were essential for some cases, hence, the number of hidden nodes incorporated in the multilayer network was larger in some simulations than others.

We observe in all Problems, except in Problem 5, that the classification rate of the model rises more rapidly than the classification rate of the multilayer network as the number of training patterns is increased. As the model approaches 100% classification rate, the multilayer network is still at low classification rates. In Problem 5, on the other hand, the multilayer network performs better than the model. We interpret the results as follows. The formal neurons that build up the multilayer network are linear separators, hence, the multilayer network is biased towards learning the training data set by means of the low order terms in

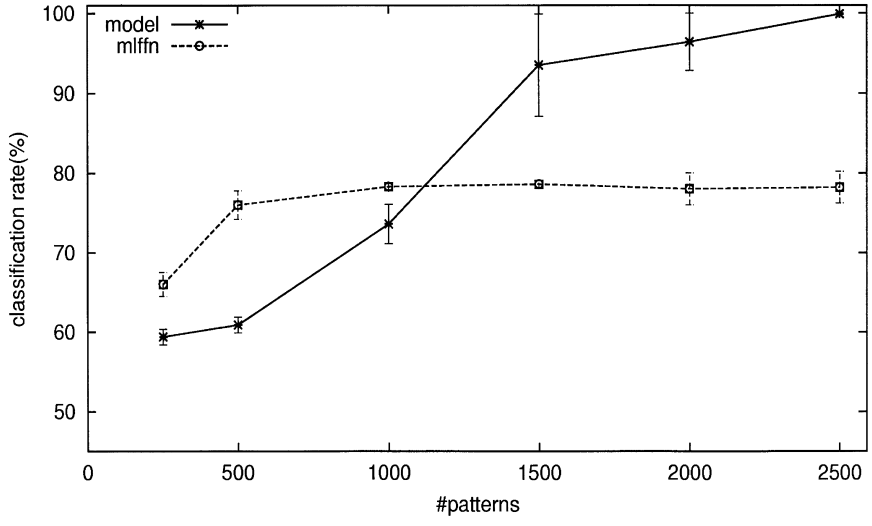

Fig. 4. Generalization for Problem 3 defined by the function $\mathrm{sgn}(1 + \prod_{j=1}^{2} x_j + \prod_{j=1}^{3} x_j + \prod_{j=1}^{19} x_j + \prod_{j=1}^{20} x_j)$ in $N = 20$ dimensions. Performance of the model is compared with the performance of the multilayer feedforward network (mlffn).

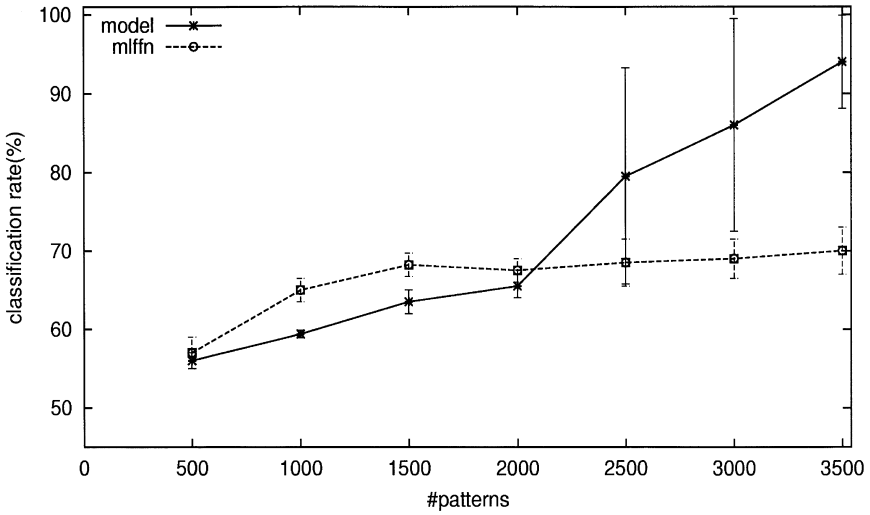

Fig. 5. Generalization for Problem 4 defined by the function $\mathrm{sgn}(1 + \prod_{j=1}^{2} x_j + \prod_{j=1}^{3} x_j + \prod_{j=1}^{4} x_j + \prod_{j=1}^{5} x_j + \prod_{j=1}^{17} x_j + \prod_{j=1}^{18} x_j + \prod_{j=1}^{19} x_j + \prod_{j=1}^{20} x_j)$ in $N = 20$ dimensions. Performance of the model is compared with the performance of the multilayer feedforward network (mlffn).

Eq. (1). On the other hand, the introduced model has no bias in favour of any term in Eq. (1). Consequently, the multilayer network performed better than the model for Problem 5 since it is a second order problem. For the other problems, on the other hand, the model performed better since those problems included some terms with order greater than two (up to order six in Problem 6, order fifteen in Problem 2, and up to order twenty in the other problems).

As seen in Fig. 2 and also in Fig. 3, learning hard problems such as the parity problem is extremely easy for the model. On the other hand, hard problems are *hard* to learn for product units; learning parity problem even at low dimensions, e.g. $N = 6$, simply does not work (Durbin & Rumelhart, 1989). This is due to the fact that, as we have foreseen, the local minima problem is not a severe problem for the model and also that the model is designed to realize learning using a minimal set; which is not the case for the product units.

Comparing Figs. 2 and 3, we observe that the model requires a greater number of training patterns in learning Problem 2 than the number of patterns in learning Problem 1. That is, to discover the redundant input channels did cost some extra number of training patterns. Problem 3 which contains terms at the highest order as well as the terms at the lowest order has been learned successfully as seen in Fig. 4. Learning Problem 3 required more patterns than learning Problem 2 and, in turn, learning Problem 4 required more patterns than learning Problem 3. This is what we expect since the number of terms that form Problem 3 is larger than the number of terms that forms Problem 2 and, in turn, the

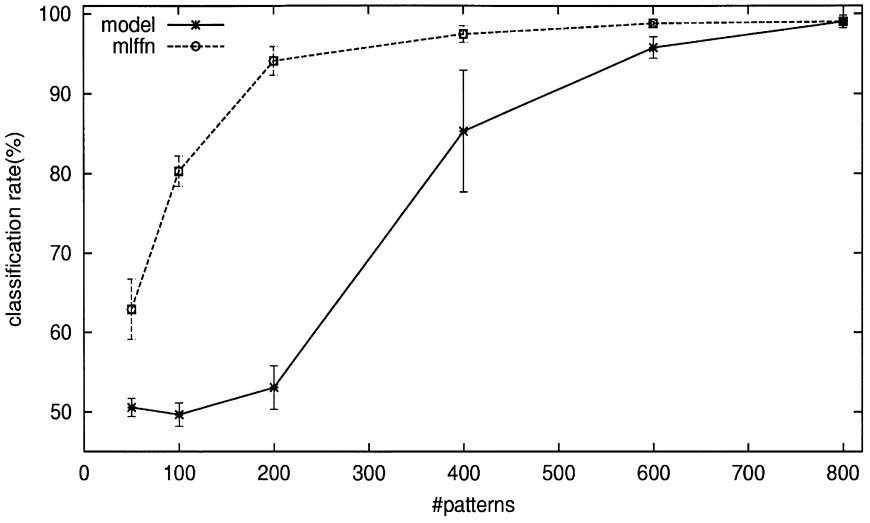

Fig. 6. Generalization for Problem 5 which is the left-right-shift problem in $N = 20$ dimensions. Performance of the model is compared with the performance of the multilayer feedforward network (mlffn).

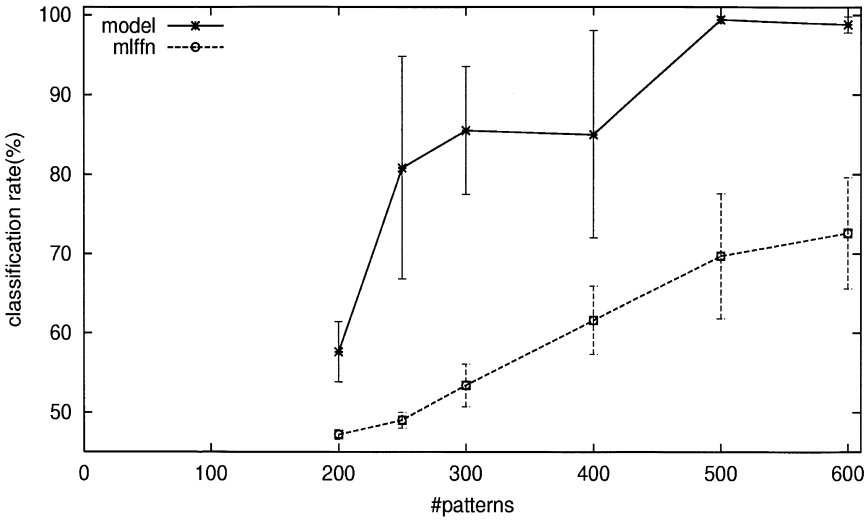

Fig. 7. Generalization for Problem 6 constructed as follows in $N = 20$ dimensions: if, in a vector, the first six bits contain an even (odd) number of -1s and the second 10 bits is obtained by the left (right) shift of the first 10 bits then the target output is $+1$, otherwise the target output is -1. Only those patterns that are well defined in the left-right-shift problem are allowed. Performance of the model is compared with the performance of the multilayer feedforward network (mlffn).

number of terms that form Problem 4 is larger than the number of terms that forms Problem 3.

Due to the weight elimination process, the result the model gives to a problem should be insensitive to the value of M provided that M is large enough so that the training data set can be learned. In order to test this, we give the generalization result for Problem 3 using $M = 15$ and $M = 30$ values in Fig. 8. As expected, the model's generalization is not sensitive to the exact value of M.

6. Discussion and conclusions

In this paper, we have introduced a model that can learn higher order correlations within the input data without suffering from the combinatorial explosion problem. This was made possible through an implicit representation of the higher order terms. The number of parameters scales as $\tilde{M} \times N$, where $\tilde{M}$ is the minimum number of higher order terms possible that can implement the input data set. The model was especially designed to realize a learning such that, after learning, the model's output for any input vector is the same as the output of a higher order network implementing the same input data set using $\tilde{M}$ number of higher order terms. The reason for such a design was that higher order networks that incorporate smaller number of terms have smaller VC dimension and, therefore, generalize better (Young & Downs, 1993). Another significant point concerning the model is that, as supported by

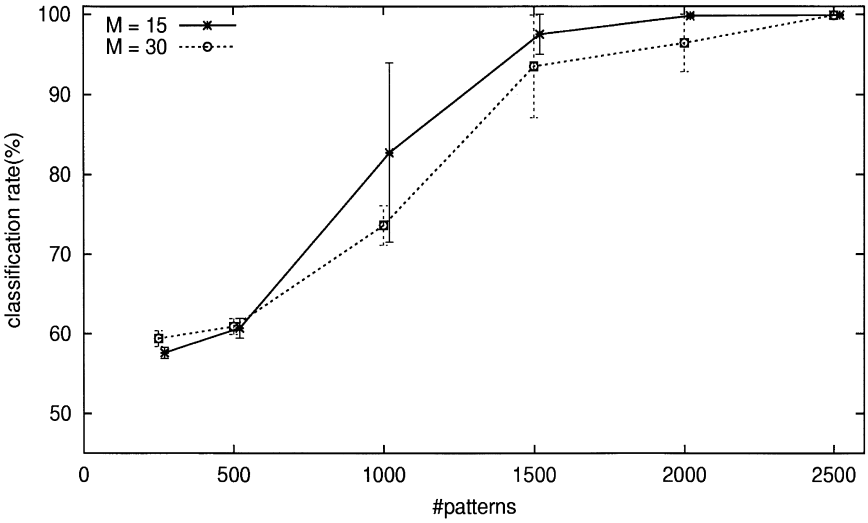

Fig. 8. Generalization by the model for Problem 3 using $M = 15$ and $M = 30$ hidden nodes.

the simulations, the local minima problem does not pose itself as a severe problem.

As demonstrated by the simulations, the model is a practical model in the sense that implementation of all patterns in the training set is always achieved by the learning algorithm simply, if necessary, by setting M to a value greater than $\tilde{M}$. Also that, if not a minimal set, a nearly minimal set will always be learned in practice. Of course, we cannot guarantee that a minimal set will always be learned since, in theory, the local minima problem is always with us even though it is not severe.

Even though the multilayer feedforward networks are universal function approximators, they are biased towards linear separability since the formal neurons are linear separators. The introduced model, on the other hand, is not biased towards low order or high order higher order terms. That is way the model gave better generalization results than the multilayer feedforward network, in the simulations, except when the problem incorporates only the low order terms. The objective of the model was just this; being able to learn higher order correlations without a priori knowledge of the order of terms or invariances. It is not unreasonable to conclude that the introduced model has an intrinsic property of learning higher order correlations.

References

Abbott, L. F., & Arian, Y. (1987). Storage capacity of generalized networks. *Physical Review A*, *36*, 5091–5094.

Baldi, P., & Venkatesh, S. S. (1987). Number of stable points for spin glasses and neural networks for higher orders. *Physical Review Letters*, *58*, 913–916.

Brady, M. J. (1990). Guaranteed learning algorithm for networks with units having periodic threshold output functions. *Neural Computation*, *2*, 405–408.

Burshtein, D. (1998). Long-term attraction in higher-order neural networks. *IEEE Transactions on Neural Networks*, *9*, 42–50.

Durbin, R., & Rumelhart, D. E. (1989). Product units: a computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, *1*, 133–142.

Fahner, G., & Eckmiller, R. (1994). Structural adaptation of parsimonious higher-order neural classifiers. *Neural Networks*, *7*, 279–289.

Gardner, E. (1987). Multiconnected neural-network models. *Journal of Physics A: Math. Gen.*, *20*, 3453–3464.

Giles, C. L., & Maxwell, T. (1987). Learning, invariances, and generalization in higher-order neural networks. *Applied Optics*, *26*, 4972–4978.

Güler, M., & Şahin, E. (1994). A binary-input supervised neural unit that forms input dependent higher-order synaptic correlations. *Proceedings of World Congress on Neural Networks*, *III*, 730–735.

Lapedes, A. & Farber, R. (1987). Nonlinear signal processing using neural networks: prediction and system modelling. (Tech. Rep. LA-UR-87-2662). Los Alamos National Laboratory, Los Alamos, NM.

Lee, Y. C., Doolen, G., Chen, H. H., Sun, G. Z., Maxwell, T., Lee, H. Y., & Giles, C. L. (1986). Machine learning using higher order correlation network. *Physica, 22D*, 276–306.

Leerink, L. R., Giles, C. L., Horne, B. G., & Jabri, M. A. (1995). Learning with product units. In G. Tesauro, D. Touretzky & T. Leen, *Advances in neural information processing systems 7* (pp. 537–544). Cambridge, MA: MIT Press.

Mel, B. W. (1990). The sigma-pi column: a model of associative learning in cerebral neocortex. CNS Memo 6, California Institute of Technology, Pasadena, California.

Perantois, S., & Lisboa, P. (1992). Translation, rotation, and scale invariant pattern recognition by higher-order neural networks and moment classifiers. *IEEE Transactions on Neural Networks*, *3*, 241–251.

Peretto, P., & Niez, J. J. (1986). Long-term memory storage capacity of multiconnected neural networks. *Biological Cybernetics*, *54*, 53–63.

Personnaz, L., Guyon, I., & Dreyfus, G. (1987). High-order neural networks: information storage without errors. *Europhysics Letters*, *4*, 863–867.

Psaltis, D., & Park, C. H. (1986). Nonlinear discriminant functions and associative memories. In J. S. Denker, *Neural networks for computing* (pp. 370–375). New York: American Inst. Physics.

Redding, N. J., Kowalczyk, A., & Downs, T. (1993). Constructive higher-order network algorithm that is polynomial time. *Neural Networks*, *6*, 997–1010.

Spirkovska, L., & Reid, M. B. (1993). Coarse-coded higher-order neural networks for PSRI object recognition. *IEEE Transactions on Neural Networks*, *4*, 276–283.

Taylor, J. G., & Coombes, S. (1993). Learning higher order correlations. *Neural Networks*, *6*, 423–427.

Weigend, A. S., Rumelhart, D. E., & Huberman, B. A. (1990). Backpropagation, weight elimination, and time series prediction. In *Proc. 1990 Connectionist Models Summer School* (pp. 65–80).

Young, S., & Downs, T. (1993). Generalization in higher-order neural networks. *Electronics Letters*, *29*, 1491–1493.