International Journal of Computational Engineering Science
Vol. 3, No. 3 (2002) 235–255

NEURAL NETWORK DECODERS FOR LINEAR BLOCK CODES

JA-LING WU
Department of Computer Science and Information Engineering
National Taiwan University,Taipei,Taiwan,R.O.C

YUEN-HSIEN TSENG
Department of Library and Information Science
Fu Jen Catholic University, Taipei, Taiwan, R.O.C.

YUH-MING HUANG
Department of Computer Science and Information Engineering
National Chi Nan University, Puli, Nantou, Taiwan, R.O.C.

This paper presents a class of neural networks suitable for the application of decoding error-correcting codes. The neural model is basically a perceptron with a high-order polynomial as its discriminant function. A single layer of high-order perceptrons is shown to be able to decode a binary linear block code with at most 2^m weights in each perceptron, where m is the parity length. For some subclass codes, the number of weights needed can be much less. The (2^m-1,2^m-1-m) Hamming code can be decoded with only m+1 weights in each perceptron. With the help of genetic algorithms, efficient neural decoders with 2t+1 terms for each bit for some t-error correctable cyclic and BCH codes are obtained. The neural decoders are formulated as a set of parity networks in the first layer followed by a linear perceptron in the second layer, and thus have simple implementations in analog VLSI technology.

1. Introduction

The error-correcting capability of a linear block code (LBC) is specified by its "minimum distance", which is the minimum Hamming distance[1] between any two codewords in the code. A systematic t-error correctable (n,k) block code is a code of minimum distance $d \geqq 2t+1$ having k information bits and n-k parity bits for each codeword. Depending on the mathematical structures imposed on a code, the class of LBCs contains the subclass of cyclic codes, which in turn contains the subclass of Bose-Chaudhuri-Hocquenghem (BCH) codes. The decoding algorithm for a t-error correctable (n,k) block codes, using syndrome-table method, requires looking up a table of size $\sum_{i=1}^{t}\binom{n}{i}$ in general. For cyclic codes, the decoding algorithm using Meggitt decodes [2] needs a table of size $\sum_{i=1}^{t}\binom{n-1}{i-1}$. The class of BCH codes has a relatively effective decoding algorithm. An algorithm of 2tn time steps has been presented in [3], although specific code can be further optimized individually.

An LBC can be described by matrix operations. Encoding of an LBC is completed by multiplying an information vector with the generator matrix. However, decoding of an LBC is generally more difficult than its encoding counterpart , because matrix inversion operation is involved .

Neural networks (NNs) are powerful computational models that have attracted much attention in many applications [4]. However, the decoding problem poses a certain level of

difficulty to most of the NN classifiers because there are a large number of categories to be classified and the classification requires very high precision so as to discriminate among patterns of only one Hamming distance from one to the others .We shall show, in this paper, that the decoding rules of a number of LBCs have a close connection with the class of high-order NNs.

Previous works on the application of NNs for decoding error control codes (ECCs) include Hopfield nets for graph-theoretic codes [4], winner-take-all networks for the (24,12) Golay code [5], Hamming nets, Counter-propagation networks (CPNs), and backpropagation networks (BPs) for Hamming codes [6,7]. The decoding of a graph-theoretic code is formulated as a problem of searching for the global minimum of a corresponding energy function of Hopfield nets. Extension to higher-order energy function for decoding LBCs has also been considered in [4]. The winner-take-all networks (called n-flop Hopfield net in [5]) together with some other housekeeping logic are applied to implement the ideal soft-decision decoder for the (24,12) Golay code. The Hamming net [6], a minimum distance classifier, decodes Hamming codes in a direct way. It stores the legal codewords in the weights of each neuron , so the number of neurons equals to the number of legal codewords . The CPN and BP decoders for (7,4) Hamming code [7] also requiring the number of hidden units equals to the number of legal codewords for reliable decoding . In other words, the complexities of the neural decoders given in [6] and [7] are in the order of 2^n.

The newly proposal approach to the decoding problem makes use of high-order perceptrons which have polynomials as their discriminant functions. The much-studied parity (exclusive-or) problem exhibits a certain level of difficulty for learning machines because similar inputs have to be classified into different categories. However, it can be shown that the parity function is isomorphic to a product term of bipolar variables. We apply high-order NNs, with the add of genetic algorithm, to decode ECCs in the hope that if they can solve the parity problem effectively, they can also solve the decoding problem without much difficulty. This is because decoding an ECC can be considered as a task of solving a set of implicity parity functions.

The main problem of using high-order polynomial discriminants is to decide what order to use and which product terms to choose. In earlier works, application of discriminant polynomials to some other problems, such as associative memories [8] - problems that can be considered as a more general case of error-correcting problems, generally requires a full set of product terms for maintaining desirable performance, (2^n weights, where n is the dimensionality of input vectors). The use of a full set of product terms in decoding ECCs does not make sense as n grows. This paper shows that the combination of high-order NNs and the simple genetic evolution is able to decode some short-length ECCs with computational cost less than those of the conventional digital decoders.

The remainder of this paper is organized as follows. In the next section, we briefly describe the model of high-order NNs and the basic idea of genetic evolution. To show the feasibility of high-order NNs model, a possible realization by reducing them to multilayer perceptrons is presented. In section 3, a single layer of high-order perceptrons

is proved to be able to decode an LBC with at most 2^m weights in each perceptron, where m is the parity length. For some subclass of codes, the number of weights needed can be much less. As a comparsion, we show that the class of single-error correcting (2^m-1,2^m-1-m) Hamming code can be decoded with only m+1 weights for each bit. This result is much better than those of the [6] and [7], in which 2^m -1 weights for each bit are required. Section 4 describes the use of genetic algorithms (GAs) to obtain better decoding structures for some cyclic and BCH codes. Finally, Section 5 presents our concluding remarks, where we indicate some possible extensions of our results.

2. Preliminaries

2.1 *The High-Order Neural Networks*

The use of polynomial discriminant function in NNs can be dated back to 1960s and the network model is called φ-machine at that time [9] and sigma-pi unit [10, 11], functional link net [12] or polynomial adaline (Padaline) [13], recently. We refer to it as high-order perceptron in this paper. The output function of a general perceptron is defined as

$$z = \mathrm{sgn}(g(x)) \tag{1}$$

In the above equation, $X = [x_1, x_2, \ldots, x_n] \in \{1,-1\}^n$ is the input pattern, 'sgn' is the sign function : sgn(a) = 1 if a > 0, -1 if a < 0, and g, the so-called discriminant function , is an rth-order polynomial:

$$g(X) = w_1 f_1(X) + w_2 f_2(X) + \ldots + w_N f_N(X) + w_0 \tag{2}$$

where w_i are called weights and each product term $f_i(x)$ is of the form: $x_{k_1}^{n_1} x_{k_2}^{n_2} \ldots x_{k_r}^{n_r}$, k_1, k_2, ... , $k_r \in \{1, \ldots, n\}$ and $n_1, n_2, \ldots, n_r \in \{0,1\}$. A linear perceptron has a discriminant function of order r = 1. A perceptron with r > 1 is called a high-order perceptron. Currently most implementation techniques for NNs cover only linear perceptrons. To make high-order perceptrons feasible, a possible realization by reducing them to multilayer perceptrons is shown below.

A product term $x_1 x_2 \ldots x_n$ of n bipolar variables is -1 if there is an odd number of -1's in the variables and +1's otherwise, which is isomorphic to the parity function of n binary elements. A two-layer linear perceptron for the parity problem has been presented in [10]. Here we reformulate the network such that its output is bipolar, weights are +1 or -1, thresholds are integers, and connection complexity is linear. Specifically, for n bipolar elements x_i, define

$$S = n - \sum_{i=1}^{n} x_i \,, \tag{3}$$

that is $S \in \{0,2,4, \ldots, 2n\}$. Let P denotes the parity function and is defined as follows:

$$P(x_1, x_2, \ldots, x_n) = 1 - \sum_{k=1}^{\left\lfloor \frac{n+1}{2} \right\rfloor} \left[\mathrm{sgn}(4k-1-S) + \mathrm{sgn}(S-(4k-3)) \right] \tag{4}$$

The value in the above square bracket is 2 if S = 2(2k-1), and 0 otherwise. So P outputs -1 if there is an odd number of -1's, and +1's otherwise. Note this is a network whose output unit needs no hard-limiting function and thus can be immediately directed to the input of its succeeding layer.

For the product of real numbers, we can write

$$X_1^{r_1} X_2^{r_2} \ldots X_n^{r_n} = \mathrm{sgn}(X_1^{r_1})\mathrm{sgn}(X_2^{r_2})\ldots\mathrm{sgn}(X_n^{r_n})|X_1^{r_1}||X_2^{r_2}|\ldots|X_n^{r_n}| \quad (5)$$

The sign part can be computed by the parity network, while the magnitude part can be recast into

$$|X_1^{r_1}|\ldots|X_n^{r_n}| = \exp\left(\sum_{i=1}^{n}\log(|X_i^{r_i}|)\right) \quad (6)$$

The operations of taking logarithm, exponential, and absolute values have simple implementations in analog VLSI technology [14]. So the high-order perceptrons, either with binary or real number inputs, can be realized using current existing techniques.

The problem of using high-order perceptrons is the combinatorial increase in the number of terms with the order of the network. It was shown in [4] that a high-order perceptron with a full set of product terms (2^n terms for n binary inputs) can dichotomize any complex training set. However, generating all combinations of n inputs becomes impractical as the input dimension grows. Thus, much effort in using high-order NNs for wider aplications focuses on determining a proper network structure for a given problem. Several approaches have been taken with satisfactory success: (i) encoding invariances into the networks [15, 16], (ii) Expressing the solution in the form representable by high-order perceptrons [17], and (iii) Using robust search algorithms to find the proper product terms. The first two approaches depend on the information available for embedding into the network structure. The third method generally requires choosing a set of terms among all possible 2^{2^n} candidates.

Once the terms are chosen, the error surface is designate to be convex with respect to the weight variables. The weights hence can be determined by the pseudo-inverse method [18] or by a suitable learning rule. The pseudoinverse method finds an optimal solution in the least-mean-square error sense between the desired output and the actual output. However it requires intensive computation that is hard to afford. An alternative approach relies on the use of a learning algorithm such as the error-correcting procedure. The error-correction procedure is the earliest learning algorithm for linear perceptrons [9]. It can be readily applied to high-order perceptrons by considering the polynomial function as a fix expansion of the original pattern space, as shown below:

$$W' = W + Y \quad \text{if } g(X) \leqq 0 \text{ and X maps to 1} \quad (7a)$$

$$W' = W - Y \quad \text{if } g(X) \geqq 0 \text{ and X maps to -1} \quad (7b)$$

where $W = [w_1, w_2, \ldots, w_N, w_0] \in R^{N+1}$ is the weight vector, W' is the new value of W, and $Y = [f_1(X), f_2(X), \ldots, f_N(X), 1] \in \{1,-1\}^{N+1}$ is the expanded input vector. This learning rule is equivalent to the delta rule $W' = W + c(d-y)Y$ at $c = 1/2$, where d is the desired output, y is the network output, and c the step size. In addition, its convergence behavior is similar to that of the delta rule: it gradually although not monotonically reduces the error energy between the actual and the desired outputs. However, the error-correcting procedures is more favorable than the notable delta rule because it involves no tuning of the step size and leads to integer weights for binary inputs, a benefit that will become clear in the later sections.

2.2 *Genetic Algorithms and Simulated Annealing*

Genetic algorithm (GA) and simulated annealing (SA) are two kinds of useful stochastic techniques which can be used to solve optimization problems efficiently. SA is based on thermodynamics and can be viewed as an algorithm which generates a sequence of Markov chains to approach the optimal solutions of the problem [19]. This sequence of Markov Chains is controlled by a gradually decreasing temperature of the system. Theoretically, the probability distribution of the system configurations generated by SA will approach to the Boltzmann distribution [19] when the system has reached equilibrium at a certain fixed temperature. When the system temperature decreases gradually to zero, the probability distribution of the system configurations generated will tend to approach the set of optimal ones. Based on this fact, SA is able to find global minimum of the optimization problem theoretically. Nevertheless, the most important parameter of SA, called system temperature, is very difficult to control. Therefore, an efficient annealing schedule is hard to design.

GAs are general purpose optimization techniques which borrow the spirit of natural selection from evolution theory. Based on natural selection, GA tries to inherit the genes with good fitness from generaton to generation. For eliminating the poor fitting candidates in each iteration, GA uses the reproduction plan to exhibit selection pressure, forcing bad ones hard to survive. In addition, GA applies genetic operations, such as crossover and mutation, to existing genes for surfing over the search space. Reproduction plans or genetic operators may behave in very different ways, but generally speaking, a typical GA often consists of a reproduction phase and a manipulation phase. The reproduction phase is responsible for exploiting the features of current candidates and reserving the well-behaved ones. The manipulation phase is reponsible for exploring the solution space and producing new possible candidates. Good GA tutorials can be found in [20, 21]. Fig. 1 shows the elementary structure of the simple genetic algorithm (SGA) [22] adopted in this paper.

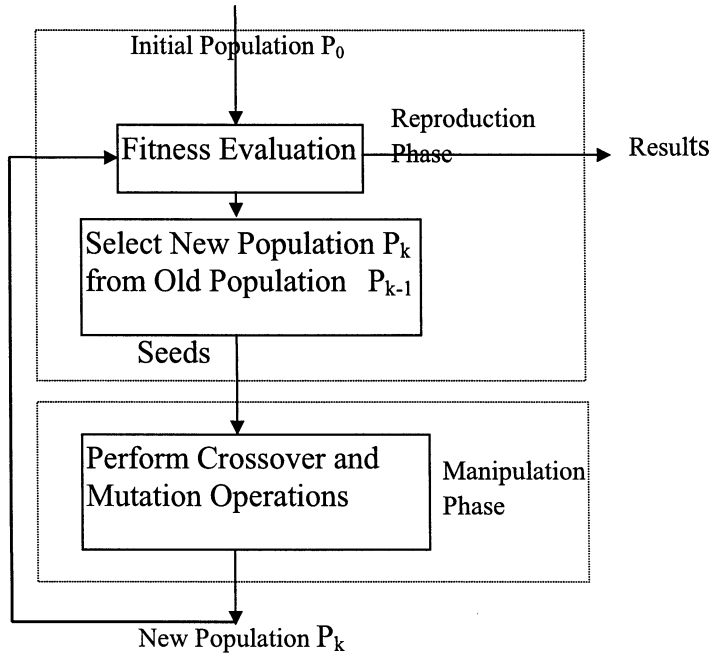

Fig 1. The basic structure of the simple genetic algorithm

3. Decoding of Linear Block Codes

The general idea of decoding an LBC can be described as follows. A systematic (n,k) code, in which there are k information bits and m = n-k parity bits for each codeword, can be represented by a parity-check matrix $H = [h_{ij}]_{m\times n}$, $h_{ij} \in \{0,1\}$. Let $A = [a_1,a_2, \ldots, a_k]$ be an information word. After A being added with some parities and then transmitted, the receiving end receives a word $V = [v_1,v_2, \ldots, v_n]$ probably with some errors in its bits. The received word V is then multiplied (mudulo2) by the parity-check matrix H to result in a syndrome vector $S = VH^T = [s_1, s_2, \ldots, s_m]$, where

$$s_i = \sum_{k=1}^{n} v_k h_{ik} \bmod 2 \tag{8}$$

The syndrome S provides the information to decode the codewords V: If S is a zero vector, there would be no errors; the parity bits v_i, i = k+1, k+2, …, n, are discarded and the information bits are directly accessed, i.e., $a_i = v_i$ for i = 1, 2, …, k. If a single error occurs in bit v_j, then S matches the jth column of H and the jth bit v_j has to be complemented. For t-error correcting codes, if S matches the sum (in modulo 2) of P (P $\leqq$ t) columns of H, then the P bits corresponding to these constituent p columns of H are in error. This decoding rule can be expressed in a Boolean formula as

$$a_j = v_j \oplus \left\{ \prod_{i=1}^{m} \neg s_i \oplus h_{ij} + \sum_{p=1}^{t-1} \left[\sum_{k_1=1}^{n} \sum_{k_2=1}^{n} \cdots \sum_{k_p=1}^{n} \left(\prod_{i=1}^{m} \neg s_i \oplus \left(h_{ij} + h_{ik1} + h_{ik2} + \ldots + h_{ikp} \bmod 2 \right) \right) \right] \right\} \quad (9)$$

for j = 1, 2, …, k, where ¬, Π, Σ, and ⊕ are NOT, AND, OR, and exclusive-OR operations, respectively. With a direct transformation from the above expression to a discriminant polynomial, we have the following theorem.

Theorem: An (n,k) binary linear block code can be decoded by a high-order perceptron with no more than 2^{n-k} terms for each bit.

Proof:

To simply the proof, let us first define two notations. Suppose a is a Boolean expression that results in a binary value and x and y are two expressions that result in a bipolar value and a real number, respectively. The notation 'a ↔x' denotes the relation between a and x as a = 1 iff x = -1, a = 0 iff x = 1, while the notation 'a ⇔ y' denotes a = 1 iff y < 0 and a = 0 iff y > 0.

Lemma 1: If $a_i \leftrightarrow x_i$ and $h_i \in \{1,0\}$ for i = 1, 2, …, n, then

$$\sum_{i=1}^{n} a_i h_i \bmod 2 \leftrightarrow \prod_{i=1}^{n} x_i^{h_j}$$

Lemma 2: IF $a \leftrightarrow x$ and $h \in \{1,0\}$ is a binary value, then $\neg a \oplus h \leftrightarrow (2h-1)x$.

Lemma 3: If $a_i \leftrightarrow x_i$ for i =1, 2, …, m, then

$$\prod_{i=1}^{m} a_i \leftrightarrow 1 - \left(\frac{1}{2}\right)^{m-1} \prod_{i=1}^{m} (1 - x_i)$$

Lemma 4: If $a_i \leftrightarrow x_i$ for i =1, 2, …, n, then

$$\sum_{i=1}^{n} a_i \leftrightarrow -1 + \left(\frac{1}{2}\right)^{n-1} \prod_{i=1}^{n} (1 + x_i)$$

By assuming that the received bit v_i and the bipolar variable x_i satisfy the relation $v_i \leftrightarrow x_i$ the syndrome element s_i in (8) can be converted into a product term t_i through lemma 1:

$$t_i = \prod_{k=1}^{n} x_k^{h_{ik}} \quad (10)$$

i =1, 2, …, m. Then applying the above lemmas, Eqn. (9) can be cast into a discriminant polynomial g_j(will be defined in (12)), each of the product term is of the form

$$X_j t_1^{r_1} t_2^{r_2} \ldots t_m^{r_m} \quad (11)$$

Because $t_i^{r_i}$ reduces to 1 if r_i is even and reduces to t_i if r_i is odd and because there are only m syndrome elements, the number of product terms in g_j never exceeds 2^m. Q.E.D.

The output of the above discrimiant g_j is strictly either 1 or -1. For certain codes, the number of terms required can be much less if the value of g_j is relaxed to be positive or negative. As an example, the (2^m-1,2^m-1-m) Hamming code can be decoded with m+1 terms in each discriminant polynomial:

$$g_j = x_j\left\{\sum_{i=1}^{m}\left[\left(2h_{ij}-1\right)t_i\right]+\left(m-1\right)\right\} \tag{12}$$

which is converted from (8) by the following lemmas:

Lemma 5: If $a_i \leftrightarrow x_i$ for i = 1, ..., m, then

$$\prod_{i=1}^{m} a_i \Leftrightarrow \sum_{i=1}^{m} x_i + \left(m-1\right)$$

Lemma 6: If $a \leftrightarrow x$ and $b \Leftrightarrow y$, then $a \oplus b \Leftrightarrow xy$.

The way of converting a decoding rule into a set of discriminant polynomials can take various forms for different codes. The Hamming code given above is an example, the Reed-Muller code presented in [23] is another. In the next section, we will present more decoding structures found by combining the high-order NN model and the SGA for some cyclic and BCH codes.

4. Decoding of Some Cyclic Codes

4.1 *The GA-based Neural Decoders*

The conventional decoding rules for cyclic and BCH codes have no direct transforms resulting in decoding structures representable by high-order perceptrons. To reduce the number of terms required, one depends on a robust optimization method and the method we choose is the genetic evolution. GAs [22, 24] are more suitable for this structure-adaptation application than other major optimization methods such as gradient-based algorithms, enumeration, and random search due to several reasons. First, the search space of possible network structures is large, 2^{2^n} possible network structures for n binary inputs. The use of enumeration method would be inefficient. Secondly, the performance surface is undifferentiable with respect to the change of used terms. Thus gradient-based searching that depends on the existence of derivatives becomes infeasible. Thirdly, different network structures can exhibit similar capabilities. Such multimodal surface would baffle most of the hill-climbing methods. Another more favorable reason for using GAs is that highly fit terms are less likely to be destroyed under the genetic operators and thus often lead to faster convergence, especially when comparing with the method of SA.

A GA begins with a randomly generated population of individuals, which are often bit-string encodings of potentail solutions to the concerned problem. Each individual is evaluated to show its fitness. High-fitness individuals receive a high selection probability for reproduction. Simple selection rule might use a roulette wheel to select an individual according to the percentage of its fitness over all population fitness. A number of genetic operators such as mutation and crossover, are applied to the selected individuals to produce new populations of the next generation. Crossover swaps the two mated individuals at a random crossing site with a crossover rate while mutation flips each bit at a predefined mutation rate. This process repeats until desired solutions emerged or a predetermined generation number has been reached.

The performance of genetic algorithms has been analyzed and has been shown that

better individuals are sampled and recombined to yield even better individuals, if the fitness function and genetic operators are properly defined. Below is the details of the network representation scheme, mutation, crossover, selection operators, and the way the fitness function is defined for the application of designing neural decoders.

As shown in Eqn. (1), a high-order neural network is a set of independent discriminant polynomials. Representation of them in a GA is quite straightforward that can take various forms depending on which factors to be optimized. We choose to save space complexity by representing a term with an integer which often take 32 bits in a machine. The presence of an input variable i in a term is represented by an one in bit i, and the absence is denoted as a zero. In such a scheme a discriminant polynomial is represented by an integer array. This representation facilitates the mutation operator by simply replacing a term with a random number ranging from 1 to 2^n-1. If there are more input variables, the integer representation of a term can be extended to an integer array, the size of which depends on the number of bits required. A discriminant polynomial is then a two-dimensional integer array. The mutation operator is slightly modified to avoid generating a product term with all bits zero in its representation.

Before presenting the crossover operator, let us examine the weights learned by the error-correction procedure in more details. Assume there are k patterns in the training set. Define a number C_i as the number of training patterns which have $f_i(X)$ the ith product term concide with their desired output, i.e., $f_i(X) > 0$ if X maps to 1 and $f_i(X) < 0$ if X maps to -1. A presentation of all the training patterns to a learning algorithm is called an iteration. If the error-correcting rule learns all the training patterns in an iteration, the weight w_i will increase by an amount of $2C_i$-k. The larger the C_i, the larger the weight w_i. Thus beginning from zero, a resultant large weight after a number of iterations implies that its corresponding product term concides with the output of most training patterns. If there are many such product terms in a discrimant polynomial, this polynomial will find earlily its weights for dichotimizing the given training set. This "larger-weight" rule leads us to a greedy method to collect the important product terms in the following crossover operation.

The crossover operator in our implementation selects a random number i and changes i terms between two mated individuals. The selected i terms in one individual to replace i terms in the other are those that do not exist in the other individual and those having larger absolute weights. If the term has already existed in the other individual, it is skipped and the term with a weight next to its weight is selected for replacing the term in the other individual of smallest weight. This process repeat until i terms are changed or no more terms can be found. As an example, suppose the two mating individuals are

$$g_1 = -7x_1+6x_1x_3-4x_2x_4+2x_1x_5-x_3x_4$$
$$g_2 = 9x_2-7x_4x_5+3x_1x_2-2x_1x_5+x_3x_5.$$

After crossover with i=2, the two new individuals would be

$$g_1' = -7x_1+6x_1x_3-4x_2x_4+9x_2-7x_4x_5$$
$$g_2' = 9x_2-7x_4x_5+3x_1x_2-7x_1+6x_1x_3.$$

Where g_1' is derived from g_1 with the last two minor terms replaced by the first two

large terms in g_2, and g_2' is derived from g_2 in a similar way. The large-weight terms corresponding to highly fit, short-defining, and low-order schemata (i.e., building blocks) which play important roles in the success of GAs. The crossover operator collects such building blocks each time it is applied. Therefore it is expected to outperform the conventional blind string swapping.

The reproduction process is implemented by merging simple roulette wheel selection and elitist model. The elitist model [22] always retains the best individual for the next generation. Although more sophisticated methods can be employed, these two simple selection rules are effective enough at the current stage.

Architecture fitness is assessed by training an individual network with the error-correction procedure and recording its performance. The error-correction procedure converges only for those linearly separable training set, but iterates endlessly for those that are not. Thus an iteration bound IB is set both to prevent from infinite loops and to obtain a rough estimation of the performance of a discriminant polynomial. Due to the rule of the genetic search, its is anticipated that smaller iteration bounds lead to more concise solutions which converge more quickly than do larger iteration ones. The fitness of the currently examined polynomial is defined as the function of the ratio C/K, where C is the number of correctly classified patterns after IB iterations and K is the number of all test patterns. Because the input patterns are bipolar, the weights learned can only take on integer values and often the weights are zeroes when they should be insignificant. Hence the error-correction procedure can sometimes annihilate a certain number of terms in the final solutions. To exploit this merit, the fitness function is modified as

$$fitness = \begin{cases} \left(\frac{C}{K}\right)^n, if \;\; C < K \\ 1 + \left(\frac{O}{N}\right)^m, otherwise \end{cases}$$

where $n > 1$, $m<1$, O is the number of zero-weights and N is the number of used terms, in the hope that the genetic selection evolves to find solutions with minimum non-zero terms. The reason for $n>1$ is to shape the fitness into a convex function, making high fitness values more prominent, and the choice for $m < 1$ favors individuals with above-one fitness values. Note that the above fitness function is still applicable for real-valued inputs if we define a zero term as the one with relatively small weight.

4.2 *Simulation Results*

The proposed neural decoder was implemented in C language on Sun Sparc-II workstation. Unless otherwise stated, a typical parameter setting chooses a crossover rate 0.9, a mutation rate 0.05, and an iteration bound IB=10. Population size and maximum generation are chosen depending on the number of training patterns. Simulation examples are taken from short-length cyclic codes. Short-length codes yield reasonable sizes of training sets and can be extended to useful long codes by various techniques in the theory of ECCs, such as product codes and nested codes [1]. They also serve as good examples to illustrate the effectiveness of the presented approach.

The first example is the class of (2^m-1,2^m-1-m) single-error correcting Hamming code. The simplest nontrivial Hamming code is the (7,4) code. It contains a total of 128 codewords, in which 16 are legal codewords and the others are composed by the 7 erroneous codewords of Hamming distance 1 from each legal ones. The four bipolar information bits are denoted as x_1, x_2, x_3, x_4 and the three parity bits are x_5, x_6, x_7. They are related as $x_5 = x_1x_2x_4$, $x_6 = x_1x_3x_4$, and $x_7 = x_2x_3x_4$. This relationship is often described by a parity matrix:

$$H = \begin{bmatrix} 1101100 \\ 1011010 \\ 0111001 \end{bmatrix}$$

where the first four columns correspond to the four information bits and the last three columns correspond to the three parity bits. So a row is the pattern indicating a parity bit and its constituent information bits.

Four independent high-order perceptrons are used to respectively decode the 4 information bits out of a received 7-bit codeword. The training set is the whole 128 possible codewords. The population size is set to 30, maximum generation is 10, and the number of used term is 20. The genetic evolution found solutions in very few generations indicating the existence of many solutions. Another phenomenon is that many solutions use only a few terms. One of the results is given below.

$$g_1(x) = 14x_1\text{-}2x_3\text{-}2x_1x_2x_5\text{+}8x_2x_4\ x_5\text{-}8x_1x_2\ x_3x_5\ x_6\ \text{-}2\ x_2x_3x_4x_5x_6\ \text{-}6x_1x_4x_5x_6x_7$$

$$g_2(x) = 12x_2\text{+}8x_3x_4x_7\text{-}4x_1x_2x_3x_4x_6\text{-}4x_2x_4x_5x_6x_7$$

$$g_3(x) = 12x_3\text{+}8x_2x_5x_6\text{+}8x_1x_4x_6\text{-}8x_1x_2x_3x_6x_7$$

$$g_4(x) = 16x_4\text{-}2x_1\text{+}8x_5x_6x_7\text{+}6x_1x_3x_6\text{-}8x_2x_3x_4x_5x_6\text{-}2$$

This result can be verified as follows. A legal codeword and its seven error patterns can be represented by

$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$$

and

$$(-x_1, x_2, x_3, x_4, x_5, x_6, x_7)$$
$$(x_1, -x_2, x_3, x_4, x_5, x_6, x_7)$$
$$(x_1, x_2,- x_3, x_4, x_5, x_6, x_7)$$
$$(x_1, x_2, x_3, -x_4, x_5, x_6, x_7)$$
$$(x_1, x_2, x_3, x_4, -x_5, x_6, x_7)$$
$$(x_1, x_2, x_3, x_4, x_5, -x_6, x_7)$$
$$(x_1, x_2, x_3, x_4, x_5, x_6, -x_7)$$

respectively. To verify the solution, one can take any of the above codewords as input to the decoder and see what the polynomial function might reduce to. Specifically, let us consider the simplest function g_3 in the above. If we present it with an erroneous word (x_1, x_2, $-x_3$, x_4, x_5, x_6, x_7) and replace the parity variables with their original constituent information variables, we have

$$\begin{aligned} g_3(x) &= -12\,x_3 + 8x_1^2x_2^2x_3x_4^2 + 8x_1^2x_3x_4^2 + 8x_1^2x_2^2x_3^3x_4^2 \\ &= -12x_3 + 8\,x_3 + 8x_3 + 8x_3 \\ &= 12\,x_3 \end{aligned}$$

The final term is x_3 and its coefficient is positive. So if x_3 =1, the perceptron produce 1, and if x_3 = -1, it produces -1; the decoder produces the correct information bit x_3 in response to the received word. Further verfication by presenting other seven codewords leads to the same result.

From the above verfication, it is observed that each term in g_i reduces to x_i after the parity variables are replaced by their constituent information variables. If a product term does not reduce to x_i, it is considered as redundant since it contributes nothing to the function g_i. This observation suggests a rule to reduce the size of the training set and the search space: Only those terms that are reducible to x_i in g_i need to be generated and the training set for these terms are a legal codeword and its correctable error patterns. Because a term containing any parity variables can always be appended with other information variables to reduce to x_i, there are at most 2^m terms to be generated, where m is the number of parity bits. Note that the threshold w_0 in the discriminant is no longer needed.

The above procedure can be applied to other longer ECCs, such as a (63,57) Hamming code. Instead of learning the complete 2^{63} codewords, only 64 patterns need to be learned and 2^6 terms need to be generated; a great reduction in the training set and the search space.

Our next example is a BCH (15,7) double error-correcting code. This code contains 8 parity bits and 120 error patterns. So the search space is 256 terms and the size of the training set is 121. The parity matrix describing this code is

$$H = \begin{bmatrix} 000101110000000 \\ 001011001000000 \\ 010110000100000 \\ 101100000010000 \\ 011101100001000 \\ 111011000000100 \\ 110011100000010 \\ 100010100000001 \end{bmatrix}$$

Solutions are found with initially 24 terms and 100 populations in 10 generations. A typical maximum and average population fitness for g_1 is shown in Fig. 2. The constant improvement in both the average population fitness and the maximum population fitness demonstrates a behavior that highly-fit terms have been accumulated and that even better solutions emerge from a population of higher-fitness individuals, a result that a random search can never exhibit.

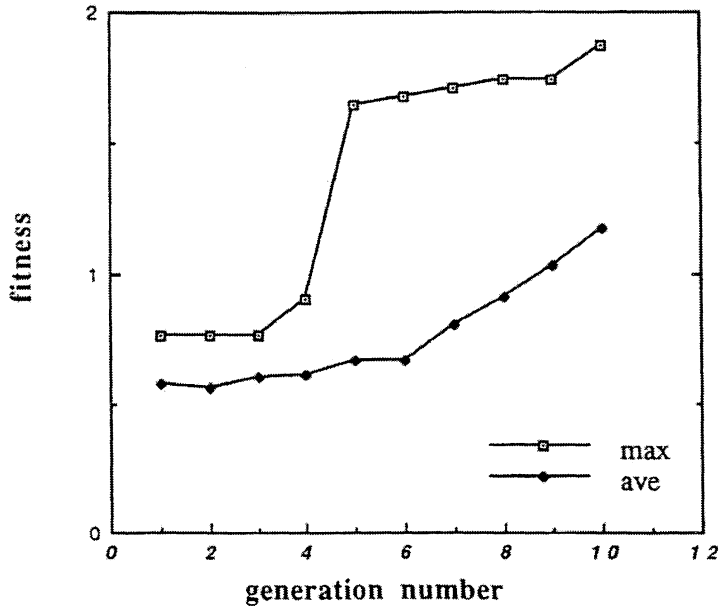

Fig. 2 Best-of-Generation Results (max) and Generation Average Results (ave) of 10 Generations for the Neural Decoder of Bit 1 of the BCH (15,7) Code.

We find in our computer outcomes that some bits can be decoded with only five positive and almost equal weights. Due to the symmetric structure of the code, all bits can be decoded with the same complexity. So we drop those terms having small weights and set the weights of the rest product terms to ones, and obtain the result:

$g_1(x) = x_1+x_3x_4x_{11}+x_5x_7x_{15}+x_2x_9x_{13}+x_8x_{10}x_{14}$

$g_2(x) = x_2+x_4x_5x_{10}+x_1x_9x_{13}+ x_6x_{14}x_{15}+x_3x_8x_{12}$

$g_3(x) = x_3+x_1x_4x_{11}+x_2x_8x_{12}+ x_7x_{13}x_{14}+x_5x_6x_9$

$g_4(x) = x_4+x_2x_5x_{10}+x_1x_3x_{11}+x_{12}x_{13}x_{15}+x_6x_7x_8$

$g_5(x) = x_5+x_2x_4x_{10}+x_{11}x_{12}x_{14}+x_1x_7x_{15}+x_3x_6x_9$

$g_6(x) = x_6+x_3x_5x_9+x_{10}x_{11}x_{13}+x_2x_{14}x_{15}+x_4x_7x_8$

$g_7(x) = x_7+x_1x_5x_{15}+x_9x_{10}x_{12}+x_3x_{13}x_{14}+x_4x_6x_8$

In the above listing, each variable occurs at most once among the five terms. Any patterns up to two bits in error (erroneous bit x_j is represented by - x_j in bipolar) lead to at most two negative terms and thus can be corrected due to majority vote (summing n bipolar bits is tantamount to determining the majority of 1 or -1).

Another a little more complicated but more delicate solution has the property that each variable occurs at most twice among the five terms and that there is a term consisting of all existing variables, that is

$g_1(x) = x_1+x_3x_4x_{11}+x_5x_7x_{15}+x_2x_9x_{13}+x_1x_2x_3x_4x_5x_7x_8x_9x_{10}x_{11}x_{13}x_{14}x_{15}$

$g_2(x) = x_2+x_4x_5x_{10}+x_1x_9x_{13}+x_6x_{14}x_{15}+x_1x_2x_3x_4x_5x_6x_8x_9x_{10}x_{12}x_{13}x_{14}x_{15}$

$g_3(x) = x_3+x_1x_4x_{11}+x_2x_8x_{12}+x_7x_{13}x_{14}+ x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{11}x_{12}x_{13}x_{14}$

$g_4(x) = x_4+ x_2x_5x_{10}+x_1x_3x_{11}+x_{12}x_{13}x_{15}+x_1x_2x_3x_4x_5x_6x_7x_8x_{10}x_{11}x_{12}x_{13}x_{15}$

$$g_5(x) = x_5 + x_2x_4x_{10} + x_{11}x_{12}x_{14} + x_1x_7x_{15} + x_1x_2x_3x_4x_5x_6x_7x_9x_{10}x_{11}x_{12}x_{14}x_{15}$$
$$g_6(x) = x_6 + x_3x_5x_9 + x_{10}x_{11}x_{13} + x_2x_{14}x_{15} + x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{13}x_{14}x_{15}$$
$$g_7(x) = x_7 + x_1x_5x_{15} + x_9x_{10}x_{12} + x_3x_{13}x_{14} + x_1x_3x_4x_5x_6x_7\, x_8x_9x_{10}x_{12}x_{13}x_{14}x_{15}$$

Thus any single and double error patterns lead to at most two negative terms and again can be surpassed by majority vote.

The BCH (15,7) code is a majority decodable code [25]. Notice the last four terms in g_1. If each of them is appended with x_1, they become

$$x_1\, x_3\, x_4\, x_{11} + x_1\, x_5\, x_7x_{15} + x_1x_2x_9x_{13} + x_1x_8x_{10}x_{14}$$
$$= t_4 + t_8 + t_2\, t_6 + t_1\, t_3\, t_7$$

where t_i is defined in (10). The GA finds exactly the same four check sums orthogonal on bit 1 as those given in [25] for the (15,7) code. Because orthogonal check sums are linear combinations of the rows of the parity matrix H, they are juxtaposition of t_i as in the form of (12) in terms of neural decoders.

Another 15-bit double-error correcting code is the (15,6) cyclic code described by the parity matrix:

$$H = \begin{bmatrix} 111001100000000 \\ 001011010000000 \\ 010110001000000 \\ 101100000100000 \\ 100001000010000 \\ 111011000001000 \\ 001111000000100 \\ 011110000000010 \\ 111100000000001 \end{bmatrix}$$

It has the similar decoding structure as the BCH (15,7) code, that is

$$g_1(x) = x_1 + x_6x_{11} + x_3x_4x_{10} + x_2x_8x_{12} + x_7x_9x_{13}$$
$$g_2(x) = x_2 + x_{10}x_{15} + x_4x_5x_9 + x_3x_7x_{11} + x_1x_8x_{12}$$
$$g_3(x) = x_3 + x_9x_{14} + x_5x_6x_8 + x_1x_4x_{10} + x_2x_7x_{11}$$
$$g_4(x) = x_4 + x_8x_{13} + x_2x_5x_9 + x_1x_3x_{10} + x_{11}x_{12}x_{14}$$
$$g_5(x) = x_5 + x_7x_{12} + x_2x_4x_9 + x_{10}x_{11}x_{13} + x_1x_{14}x_{15}$$
$$g_6(x) = x_6 + x_1x_{11} + x_3x_5x_8 + x_9x_{10}x_{12} + x_2x_{13}x_{14}$$

Each variable occurs at most once, so it can correct any single and double error patterns.

Due to above decoding structures, we conjecture that any t-error correcting majority decodable code has the decoding structure of 2t+1 terms for each bit in the neural decoder. Note that to make this possible the terms must contain as little variables as possible so as to keep the occurrence of each variable low. One good candidate for such terms in g_i is x_i. The others can be searched by GA with a bias toward low-order terms when they are randomly generated in the beginning. After incorporating this information into our work, we are able to search the decoding structure of the previous two codes more quickly and of some other codes which having larger search space.

The BCH (15,2) code has a minimal distance 10 and is four-error correctable and five-error detectable. The search space contains 8192 terms and the training set is of 1941 patterns, quite a large search space and training size. The following solution emerges in 10 generations with 100 populations and initially 36 terms. The corresponding discriminant functions can be described as :

$$g_1(x) = x_1 + x_5 + x_8 + x_{11} + x_{14} + x_2 x_3 + x_6 x_7 + x_4 x_9 + x_{12} x_{13} + x_{10} x_{15}$$

$$g_2(x) = x_2 + x_4 + x_7 + x_{10} + x_{13} + x_3 x_8 + x_1 x_9 + x_6 x_{11} + x_{12} x_{14} + x_5 x_{15}$$

where the parity matrix of this code is

$$H = \begin{bmatrix} 111000000000000 \\ 010100000000000 \\ 100010000000000 \\ 110001000000000 \\ 010000100000000 \\ 100000010000000 \\ 110000001000000 \\ 010000000100000 \\ 100000000010000 \\ 110000000001000 \\ 010000000000100 \\ 100000000000010 \\ 110000000000001 \end{bmatrix}$$

Again each bit variable presents exactly once. So the 10 terms can correct any patterns up to four bits in error and detect the existence of those five-bit errors for which case at least one of the discriminant polynomials is zero. Because a single term can correct even errors of its containing variables , this decoder can correct some patterns of more than four bits in error. Although it is not listed in [25] as a majority decodable code, the above result confirms that BCH (15,2) can be decoded by majority logic.

Table 1 lists some cyclic codes and their corresponding number of terms required for decoding. To specify a code, the code length n, number of information bits k, the minimum distance d, the minimum distance guaranteed by the BCH bound d_{BCH}, and the exponents of the roots of the generator polynomial are tabulated like that in [2]. As the table listed, some t-error correctable cyclic codes can be decoded with 2t+1 terms (or 2t+2 terms for (t+1)-error detection at the same time). These are good decoding structures. Some of them have been shown as majority decodable in the literature, while some of them have not to our best knowledge. For other codes requiring more terms, the listed numbers in the table are the least ones we obtain so far. Their decoding structures are different from the previous ones. For example, the (21,7) code with d = 8 and d_{BCH} = 5 uses 14 terms of various weights: positive, negative, large, and small, making it hard to

interpret the structure it represents. Nevertheless, more simulations using other domain knowledge of ECC and other heuristic rules may lead to even less terms for correct decoding.

4.3 *Comparison with Digital Decoders*

As prescribed, the decoding algorithm for a t-error correctable (n,k) blocks codes using syndrome-table method generally requires looking up a table of size $\sum_{i=1}^{t}\binom{n}{i}$. For cyclic codes, the decoding algorithm using Meggitt decoders needs a table of size $\sum_{i=1}^{t}\binom{n-1}{i-1}$. The class of BCH codes has a relatively effective decoding algorithm. However, its error-correction capability is limited by the BCH bound. That is, if a code is designed to correct t_0 error, in some cases it may have a minimum distance $d = d_{BCH} > 2t_0+1$; that is not all correctable errors can be corrected by the algorithm. One example is the (21,7) code presented in Table 1. Thus the following comparison is made between a Meggitt decoder and a neural decoder.

Consider Fig. 3 the Meggitt decoder for an (n,k) cyclic code [1]. The upper half is a division circuit, which performs shift-and-add (in modulo 2) operations. The lower half consists of a comparison circuit and an n-bit shift register. After n shifts to receive an n-bit codeword, the comparator has to simultaneously match the result of the division circuit with $\sum_{i=1}^{t}\binom{n-1}{i-1}$ patterns in each of the next n shifts to correct the received word. The complexity of this decoder mainly lies in the comparator. Figure 3(b) depicts a neural decoder for the same (n,k) code. Once an n-bit codeword is received, the k clean

n	k	d	d_{BCH}	Roots of Gen. Poly	No. of Terms
15	7	5	5	1,3	5
15	6	6	6	0,1,3	5
15	5	7	7	1,3,5	25
15	4	8	8	0,1,3,5	8
15	3	5	5	1,3,7	5
15	2	10	10	0,1,3,7	10
17	8	6	6	0,1	14
21	12	5	5	1,3	23
21	11	6	6	0,1,3	5
21	10	5	5	1,3,7	19
21	9	6	5	1,3,9	5
21	8	6	6	0,1,3,9	10
21	8	6	6	0,1,5	12
21	7	8	5	1,3,7,9	14
21	6	8	6	0,1,3,7,9	11
21	6	7	7	1,3,5	12

Table 1. Some Cyclic Codes and Their Corresponding Number of Terms Required for Decoding.

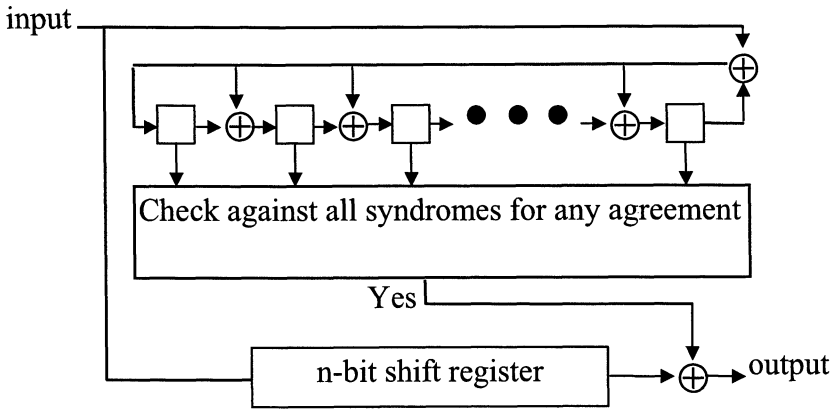

Fig. 3 (a)

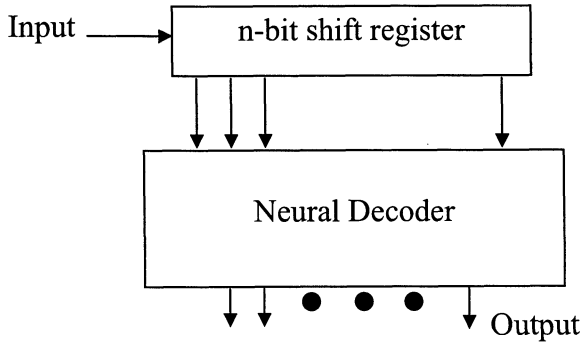

Fig. 3 (b)

Fig. 3 (a) A Meggit Decoder, (b) A Neural decoder

information bits are available by decoding them in parallel using k discriminant polynomials. It is not necessary to force the input to be received sequentially as shown in Fig. 3(b); the input may come in parallel.

As shown in Section 2.1, a product term of bipolar variable in a discriminant polynomial can be implemented by a parity network of linear connection complexity. The neural decoder thus can be modeled as a multilayer perceptron with a number of parity networks in the first-half layer, followed by a linear perceptron in the second-half layer, which make its realization simple in analog VLSI circuits or other abundant technologies [26-29].

Apparently, the neural decoder with the structure of using 2t+1 terms uses less

hardware resources. The other neural decoders using a little more terms might have this advantage as well after minimizing the decoders by extracting common terms among the discriminant polynomials.

5. Discussions and Conclusion

Since the population size and the maximum generation number of the proposed genetic evolution are chosen depending on the number of required training patterns, which is in the order of 2^{n-k} for an (n,k) cyclic code in general. Simulation examples shown in the previous section were taken from short-length cyclic codes to yield reasonable size of training sets. In other words, efficient decoding structures of long length codes that can be found by the proposed approach are still limited by the available memory size and the affordable computation time. There are two ways to expand the practical value of the proposed approach to find larger length codes :

First, longer codes can be constructed from the codes of Table 1 by the techniques of interleaving. To get a (bn,bk) code from an (n,k) code, taking any b codewords from the original code and merge the codewords by alternating the symbols. If the original code can correct any burst error of length t, it is apparent that the interleaved code can correct any burst error of length bt. For example, by taking five copies of the (21,11) code and interleaving the bits, one obtains a (105,55) code. Because each of the individual codes can correct a burst error of length two, the new code can correct any burst error of length ten.

Instead of the coding theory itself, the second way to expand the capability of the proposed decoder relies on including more powerful evolutionary computation algorithms into our work. For example, one can combine GA and SA together to construct the so-called Annealing-Genetic algorithms (AGAs) which have successfully been applied for solving large size optimization problems, such as the Traveling Salesman [30] and the Molecular Binding [31]. Of course, the whole utilization of the power of AGA in searching for efficient decoding structures depends heavily on whether one can integrate proper ECC domain-knowledge with the design and selection of annealing schedule and genetic evolution parameters or not. There are a lot of works need to be done before any efficient decoding structure of larger codes can be reported. This will certainly be the major direction of our future work.

In this paper, we have shown a possible realization of high-order neural networks by reducing them to multilayer perceptrons. From a practical point of view, this reveals that a high-order neural network can be implemented as easy as (or as hard as) a multilayer perceptron can be. The high-order neural networks proposed here might even have a simpler implementation since they use a simple learning algorithm, error-correcting procedure, instead of a more complicate one, for example, back-propagation used by multilayer perceptrons. We have also shown that a syndrome decoder or a one-step majority decoder can be converted directly to high-order neural network with 2^{n-k} or $2t+1$ product terms, respectively. This is an expected result since high-order neural networks have been shown to be extremely effective to the parity (exclusive-or) problem.

With the help of a genetic-type search algorithm, we have shown that a suitable set of product terms of a high-order neural network for a given problem (error correcting

decoder in this case) can be found. It should be noticed that this approach can not only be applied to a structured problem as we discussed here but also to a some what unstructured problem [32]. The proposed genetic-neural approach has found the majority decoders (in their equivalent forms of high-order neural networks) for some linear codes that have not been known as majority decodable. In other words, this algorithm can also serve as a searching algorithm for majority decodable codes and the majority decoding equations for them.

For neural network hardware containing analog components, as pointed out by one of the reviewers, the dynamic reconstruction and on-line parameter estimation is not an easy task. In other words, some sensitivity analyses have to be done before an appropriate silicon implementation can be accomplished. However, VLSI implementation of the proposed decoder is not the main focus of this paper, and therefore, this valuable subject will be one of our future research topics.

References

1. R. E. Blahut, Theory and Practice of Error Control Codes, Addison-Wesley, 1983.
2. W. Wesley Peterson and E. J. Weldom, Jr. Error-Correcting Codes, 2nd ed., Cambridge, MIT Press, 1972.
3. R. E. Blahut, "A Universal Reed-Solomon Decoder" IBM J. Res. Develop., Vol. 28, No. 2, March 1984, pp. 150-158.
4. J. Bruck and M. Blaum, "Neural Networks, Error-Correcting Codes, and Polynomails Over the Binary N-Cube", IEEE Trans. on Information Theory. Vol.35, pp. 976-987, 1989.
5. Jing Yuan and C. S. Chen, "Correlation Decoding of the (24,12) Golay Code Using Neural Networks", IEE Proceedings-I, Vol. 138, No. 6, pp. 517-524, Dec. 1991.
6. G. Zeng, D. Hash, and N. Ahmed, "An Application of Neural Net in Decoding Error-Correcting Codes", IEEE, Int. Sym. on Circuits and Systems, 1989, pp. 782-785.
7. A. D. Stefano and O. Mirabella, "On the Use of Neural Networks for Hamming Coding", IEEE, Int. Sym. on Circuits and Systems, 1991 pp. 1601-1604.
8. X. Xu, and T. Tsai, "Constructing Associative Memories Using Neural Networks," Neural Networks, Vol. 3, pp. 301-309, 1990.
9. Nils J. Nilsson, Learning Machines : Foundations of Trainable Patter-Classifying Systems, McGraw-Hill, 1965.
10. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation" in Parallel Distributed Processing, D. E. Rumelhart and PDP Research Group, Cambridge, MA:MIT Press 1986.
11. R. Durbin and D. E. Rumelhart, "Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks", Neural Computation, Vol. 1, No. 1, pp. 133-143, 1989.
12. Yoh-Han Pao, Adaptive Pattern Recognition and Neural Networks, Addison-Wesley, 1989.
13. Donald F. Specht, "Probabilistic Neural Networks and the Polynomial Adaline as

Complementary Techniques for Classification", IEEE Trans. on Neural Networks, Vol. 1, No. 1, March, pp. 111-121, 1990.
14. Carver A. Mead, Analog VLSI and Neural Systems, Reading, MA: Addison-Wesley, 1989.
15. C. L. Giles and T. Maxwell, "Learning, Invariance, and Generalization in High-Order Neural Networks", Applied Optics, Vol. 26, No. 23, pp. 4972-4978, Dec. 1987.
16. S. J. Perantonis and P. J. G. Lisboa, "Translation, Rotation, and Scale Invariant Pattern Recognition by High-Order Neural Networks and Moment Classifiers," IEEE Trans. on Neural Networks, Vol. 3, No. 2, March 1992, pp. 241-251.
17. Yuen-Hsien Tseng and Ja-Ling Wu, "Solving Sorting and Related Problems by Quadratic Perceptrons", Electronics Letters, Vol. 28, No. 10, pp. 906-908, 1992.
18. T. Kohonen, Self-Organization and Associative Memory, Springer-Verlag, 1987.
19. P. J. M. Van Laarhoven, E. H. L. Aarts, Simulated Annealing: Theory and Applications, D. Reidel Publishing Company, Dordrecht, Holland, 1987.
20. D. E. Goldberg, "Genetic and Evolutionary Algorithms Come of Age," Communication ACM, Vol. 37, No. 2, pp. 113-119, Feb. 1994.
21. M. Strinivas, L. M. Patnaik, "Genetic Algorithms : A Survey," IEEE Computer Magazine, pp. 17-26, 1994.
22. D. E. Goldberg, Genetic Algorithms : In Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, 1989.
23. Yuen-Hsien Tseng and Ja-Ling Wu, "Decoding Reed-Muller Codes by Multilayer Perceptrons," the International Journal of Electronics, vol. 75, No. 4, pp. 589-594, 1993.
24. J. H. Holland, Adaptation in Natural and Artificial Systems, Ann Arbor: The University of Michigan Press, 1975.
25. Shu Lin and Daniel J. Costello, Jr., Error Control Coding : Fundamentals and Applications, Prentice-Hall, 1983.
26. F. J. Kub, K. K. Moon, I. A. Mack, and F. M. Long, "Programmable Analog Vector-Matrix Multipliers," IEEE Journals of Solid-State Circuits, Vol. 25, No. 1, pp. 207-214, 1990.
27. J. B. Lont and W. Guggenbuhl, "Analog CMOS Implementation of a Multilayer Perceptron with Nonlinear Synapses," IEEE Trans. on Neural Networks, Vol. 3, No. 3, pp. 457-465, May 1992.
28. L. W. Massengill and D. B. Mundie, "An Analog Neural Hardware Implementation Using Charge-Injection Multiplier and Neuron-Specific Gain Control," IEEE Trans. on Neural Networks, Vol. 3, No. 3, pp. 354-362, May 1992.
29. G. Moon, M. E. Zaghloul and R. W. Newcomb, "VLSI Implementation of Synaptic Weighting and Summing in Puse Coded Neural-Type," IEEE Trans. on Neural Networks, Vol. 3, No. 3, pp. 394-403, May 1992.
30. F. T. Lin, C. Y. Kao, and C. C. Hsu, "Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems," IEEE Trans. on System, Man, Cybernetics, Vol. 23, No. 6, pp. 1752-1767, 1993.

31. L. H. Wang, C. Y. Kao, M. Ouh-young and W. C. Chen, "Molecular Binding : A Case Study of the Population-based Annealing Genetic Algorithms," IEEE Inte'l Conf. on Evolutionary Computing, Perth, Australia 1995.
32. Yuen-Hsien Tseng and Ja-Ling wu, "Constructing Associative Memories by High-Order Neural Networks", Electronics Letters, Vol. 28, No. 12, pp. 1122-1124, 1992. Also appears in Asia Pacific Conference on Circuits and Systems, 1992, Australia.