Pattern Recognition Letters 19 (1998) 559–568

Pattern Recognition Letters

Perceptron–perceptron net

Sheng-De Wang [*], Tsong-Chih Hsu

Department of Electrical Engineering, EE Building, Rm. 441, National Taiwan University, Taipei, Taiwan, ROC

Received 5 February 1997; revised 20 February 1998

Abstract

It is well known that if a Boolean function is expressed in sum of products, each function can be implemented with one level of AND gates followed by an OR gate. We will prove that if each desired output of a binary function is expressed in sum of products, each desired output can be implemented with one layer of perceptron nodes followed by a perceptron node.
© 1998 Elsevier Science B.V. All rights reserved.

Keywords: Functional link networks; Binary mapping; Dimension expansion

1. Introduction

The multilayer feedforward structure with the backpropagation (BP) training algorithm (Rumelhart et al., 1986), has been one of the most widely used artificial neural network models and can be used to approximate any mapping. Suppose we want to approximate a particular set of functions to a given accuracy; one thing is to determine the number of hidden layers and the number of units per layer we need. The answer is at most two hidden layers, with arbitrary accuracy being obtainable given enough units per layer (Cybenko, 1988). It has also been proven that only one hidden layer is enough to approximate any continuous function (Cybenko, 1988; Hornik et al., 1989). The utility of these results depends on how many hidden units are necessary, and this is not known in general. On the other hand, we may think about how to use binary neural networks to realizing discrete functions. It is well known that if a Boolean function is expressed in sum of products, it can be implemented with one level of AND gates followed by an OR gate. We would like to know what is the case for neural networks. In this letter, we will prove that if each desired output of a binary function is expressed in sum of products, each desired output can be implemented with one layer of perceptron nodes followed by a perceptron node.

It is well known that the simple perceptron algorithm (Rosenblatt, 1958) is unable to represent classifications which are not linearly separable (Minsky and Papert, 1988). However, the perceptron does exhibit two very useful properties. First, the perceptron learning algorithm will either converge or will cycle among a series of hyperplanes that do not fully separate the input patterns. Second, the topology of the network is not a design

[*] Corresponding author.

issue since there are no hidden layers. To extend perceptrons to solve the linearly nonseparable problem, one concept is to find a suitably enhanced representation of the input data. The enhanced input vectors are composed of original input vectors plus some expanded dimension. Methods such as the dimension expansion method (Section 2) and functional link networks (next subsection) make use of this concept to solve the linearly nonseparable problem.

1.1. Review of functional link networks

The block diagram of a functional link network (Pao, 1989) is shown in Fig. 1. The main idea of the functional link network is to find a suitably enhanced representation of the input patterns. The so-called *tensor model* is suitable for handling input patterns in the form of vectors. Assume the original input patterns are represented by n-tuple vectors, $x_1, \dots, x_n$. The higher-order input terms are obtained as the products $x_i x_j$ for all $1 \leqslant i,j \leqslant n$ and $i < j \leqslant n$, the products $x_i x_j x_k$ terms for all $1 \leqslant i,j,k \leqslant n$ and $i < j < k \leqslant n$, $\dots$, and the products $x_{i_1} x_{i_2} \cdots x_{i_{n-1}}$ terms for all $1 \leqslant i_1, i_2, \dots, i_{n-1} \leqslant n$ and $i_1 < i_2 < \cdots < i_{n-1} \leqslant n$. Thus we have

$$\binom{n}{2} + \binom{n}{3} + \cdots + \binom{n}{n-1} = 2^n - n - 1$$

higher-order input terms. The extremely large number of higher-order terms sometimes makes the approach impractical. In the so-called *functional model* of a functional link network, the higher-order input terms may be generated using the orthogonal basis functions. Functions such as $\sin(\pi x)$, $\cos(\pi x)$, $\sin(2\pi x)$, $\dots$, can be used to enhance the representation of input x.

This letter will present a systematic method for the design of Binary Neural Networks (BNNs). Our method (Section 2) is similar to the functional link network. In our method, we use the training pairs (include input patterns and desired patterns) to obtain the higher-order input terms. Our method is guaranteed to converge to a set of solution weights of the problem in finite time.

1.2. Previous work

There are many algorithms for modeling BNN in the past decade. Moody and Antsaklis (1996) propose an algorithm to construct and train multilayer neural networks. A training algorithm for four-layer perceptron-type binary feedforward neural networks was presented by Gray and Michel (1992) and was used on a feedforward neural network for the generation of binary-to-binary mapping. The oil-spot algorithm by Mascioli and

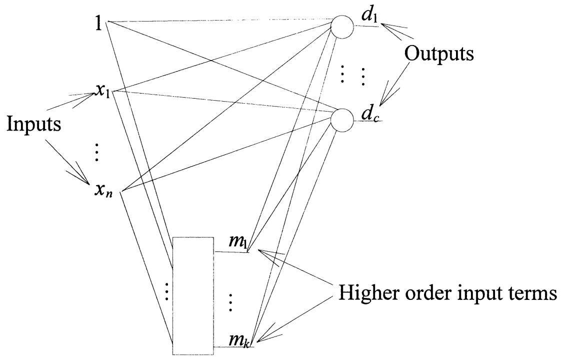


Fig. 1. Functional link network.

Martinelli (1995) relies on a topological approach to solve the problem. The upstart algorithm (Frean, 1990) constructs a tree of nodes to enrich the possibility of finding a solution. A binary classification method proposed by Nadal (1989) trains the perceptron with the pocket algorithm to find the best set of weights and cascades the output unit with a new unit that receives signals from inputs and the previous output unit. The algorithm runs until all patterns are classified correctly. The Gallant's Tower (Gallant, 1986) indicates that it adds new output node that also receives input from previous output node, and continue training until the solution is satisfied. All these methods are based on a multilayer structure.

To solve the problem of binary mapping with a simpler neural structure, we propose using the perceptron–perceptron net. The proposed method extends the set of inputs of a binary input, single binary output neural network in such a way that the resulting network can solve non-linearly separable problems, yet, can still be trained with the perceptron learning algorithm, and yet, can be implemented with one layer of perceptron nodes followed by a perceptron node.

The rest of this letter is organized as follows. Section 2 will describe the proposed dimension expansion method, a theorem and its proof, and an example to illustrate the ideas. Section 3 will describe the adaptive dimension expansion method. Finally, conclusions are presented.

2. Dimension expansion method

2.1. Review of single-layer binary bipolar perceptron networks

A perceptron neuron, which has a hard limit transfer function, is shown in Fig. 2. In Fig. 2, the output of the perceptron, y, is given by

$$y = f(\text{net}),$$

where

$$\text{net} = W'X = X'W = \sum_{i=0}^{n} w_i x_i, \quad X = [x_0\ x_1\ \dots\ x_n]' \quad \text{and} \quad W = [w_0\ w_1\ \dots\ w_n]'.$$

Each external input X, augmented with the bias $x_0 = 1$, is weighted with an appropriate W and the sum of the weighted inputs is sent to the hard limit transfer function. The transfer function, $f(\cdot)$, returns a 0 if $W'X < 0$ or

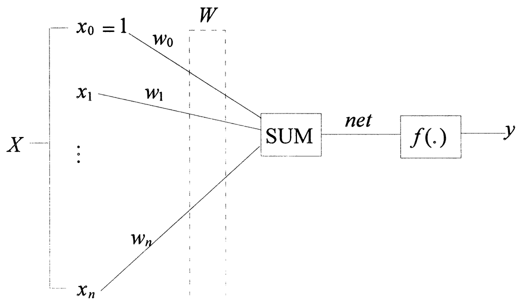


Fig. 2. A perceptron network.

a 1 if $W'X \geqslant 0$. The hard limit transfer function enables a perceptron to classify input vectors by dividing the input space into two regions. The output will be either 0 or 1, depending on the classification of the input. Let us concentrate on the output y of the neuron. If we present a batch of p input vectors ($X^{(i)} = [x_0^{(i)}\ x_1^{(i)}\ \dots\ x_n^{(i)}]'$, where $i = 1,\dots,p$) to this network, then we have the sequence of outputs $y^{(i)}$, where $i = 1,\dots,p$,

$$\begin{aligned} y^{(1)} &= f\big(W_0 + w_1 x_1^{(1)} + w_2 x_2^{(1)} + \cdots + w_n x_n^{(1)}\big),\\ y^{(2)} &= f(w_0 + w_1 x_1^{(2)} + w_2 x_2^{(2)} + \cdots + w_n x_n^{(2)},\\ &\vdots\\ y^{(p)} &= f\big(w_0 + w_1 x_1^{(p)} + w_2 x_2^{(p)} + \cdots + w_n x_n^{(p)}\big). \end{aligned} \tag{1}$$

In order to analyze further, we first try to remove the transfer function, $f(\cdot)$. Given are p training pairs $\{X^{(i)},d^{(i)}\}$, $i = 1,2,\dots,p$, where $X^{(i)} = [1\ x_1^{(i)}\ \dots\ x_n^{(i)}]$ is $(n+1)\times 1$ and the desired output $d^{(i)}$ is 1×1. Note that a sufficient condition for (1) to hold in the training phase is that the linear actual outputs, $y^{(i)}$, is equal to the desired outputs

$$\begin{aligned} d^{(1)} &= w_0 + w_1 x_1^{(1)} + w_2 x_2^{(1)} + \cdots + w_n x_n^{(1)},\\ d^{(2)} &= w_0 + w_1 x_1^{(2)} + w_2 x_2^{(2)} + \cdots + w_n x_n^{(2)},\\ &\vdots\\ d^{(p)} &= w_0 + w_1 x_1^{(p)} + w_2 x_2^{(p)} + \cdots + w_n x_n^{(p)}. \end{aligned} \tag{2}$$

This condition will be used in the following sections for the discussion of the dimension expansion. In general, we describe equations in the matrix notation form,

$$XW = D,$$

where

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)}\\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & x_1^{(p)} & x_2^{(p)} & \cdots & x_n^{(p)} \end{bmatrix},\quad W = \begin{bmatrix} w_0\\ w_1\\ w_2\\ \vdots\\ w_n \end{bmatrix} \quad\text{and}\quad D = \begin{bmatrix} d^{(1)}\\ d^{(2)}\\ \vdots\\ d^{(p)} \end{bmatrix}. \tag{3}$$

We want to know whether there exist solutions to $XW = D$, and if so, how to find the solutions. In linear algebra, there are two theorems that can be applied to our equations: (I) Gauss elimination and solution sets: Suppose that the system of equations $XW = D$ – or, equivalently, the augmented matrix $[X\ D]$ – is transformed by a sequence of elementary row operations into the system $\tilde{X}W = \tilde{D}$ – or, equivalently, into the augmented matrix $[\tilde{X}\ \tilde{D}]$. Then the solution sets are identical: W solves $XW = D$ if and only if W solves $\tilde{X}W = \tilde{D}$. (II) Rank and solvability: For the system of equations $XW = D$, exactly one of the following three possibilities will hold:

1. The system $XW = D$ has no solution if and only if the rank of the augmented matrix $[X\ D]$ is greater than that of X.
2. The system $XW = D$ has exactly one solution if and only if the rank of $[X\ D]$ equals that of X, and equals the number of unknowns.
3. The system $XW = D$ has infinitely many solutions if and only if the rank of $[X\ D]$ equals that of X, and is strictly less than the number of unknowns.

2.2. Dimension expansion

We know that perceptrons can only classify linearly separable sets of vectors. If a straight line or a hyperplane can be drawn to separate the input vectors into their correct categories, the given classes are linearly separable. If the given classes are not linearly separable, the learning process will never reach a point where all given classes are classified properly. The dimension expansion method is to make the given classes linearly separable before doing the perceptron learning procedure. Each external input $[x_0\ x_1\ \dots\ x_n]'$ is expanded to $[x_0\, x_1\ \dots\ x_n\ x_{n+1}\ x_{n+2}\ \dots\ x_{n+k}]'$, where $k = p - 1$, for $p \geqslant 1$. We can describe the expanded equations using the following matrix notation:

$$UW = D,$$

where

$$U = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} & x_{n+1}^{(1)} & \cdots & x_{n+k}^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} & x_{n+1}^{(2)} & \cdots & x_{n+k}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_1^{(p)} & x_2^{(p)} & \cdots & x_n^{(p)} & x_{n+1}^{(p)} & \vdots & x_{n+k}^{(p)} \end{bmatrix}, \quad W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} d^{(1)} \\ d^{(2)} \\ \vdots \\ d^{(p)} \end{bmatrix}. \tag{4}$$

We propose the expanded matrix U as below:

$$U = [X \quad H] = \left[\begin{array}{ccccc:cccc} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} & 0 & & \mathbf{O} & \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} & 1 & \ddots & & \\ \vdots & \vdots & \vdots & \ddots & \vdots & & 1 & \ddots & \\ 1 & x_1^{(p)} & x_2^{(p)} & \cdots & x_n^{(p)} & & \mathbf{O} & & 1 \end{array}\right]. \tag{5}$$

$$\overbrace{}^{H} \qquad \underbrace{}_{p-1}$$

Lemma 1. *It is assumed that there are more training patterns than inputs, i.e., $p > n$. The rank of U in (5) is equal to the rank of $[U\ D]$.*

Proof. Since only one element of any column of H is 1 and the others are all 0, the rank of H is equal to $p - 1$. since $p < (n+1) + (p-1)$, the rank of U is equal to p. Since $p < (n+1) + (p-1) + 1$, the rank of $[U\ D]$ is equal to p. □

So, the rank of U is equal to the rank of $[U\ D]$, and thus the system of equations $UW = D$ has exactly one solution or has infinitely many solutions. This shows that if we expand the dimension of input space as (5), then there exist solutions for the system, i.e., suitable weights can be obtained by an appropriate learning algorithm.

Lemma 2. *Training pairs with input X (in (3)) and output H (in (5)) have perceptron solutions.*

Proof. Training pairs are $\{X^{(i)}, H^{(i)}\}$, where $H^{(i)} = [h_1^{(i)}\, h_2^{(i)}\ \dots\ h_k^{(i)}]'$ is the transpose of the ith row of H as

Table 1
A 2-input and 3-output truth table

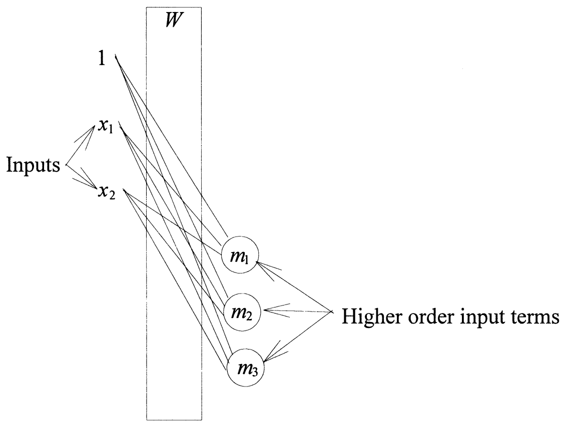
x_1	x_2	d_1	d_2	d_3
0	0	0	0	1
0	1	1	0	0
1	0	1	1	1
1	1	0	1	0

shown in (5) ($i = 1,2,\dots,p$), and only one element of any column of H is 1 and the other elements are all 0. In general, let us consider the augmented matrix:

$$\left[\begin{array}{ccccc:c} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(i-1)} & x_2^{(i-1)} & \cdots & x_n^{(i-1)} & 0 \\ 1 & x_1^{(i)} & x_2^{(i)} & \cdots & x_n^{(i)} & 1 \\ 1 & x_1^{(i+1)} & x_2^{(i+1)} & \cdots & x_n^{(i+1)} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(p)} & x_2^{(p)} & \cdots & x_n^{(p)} & 0 \end{array}\right].$$

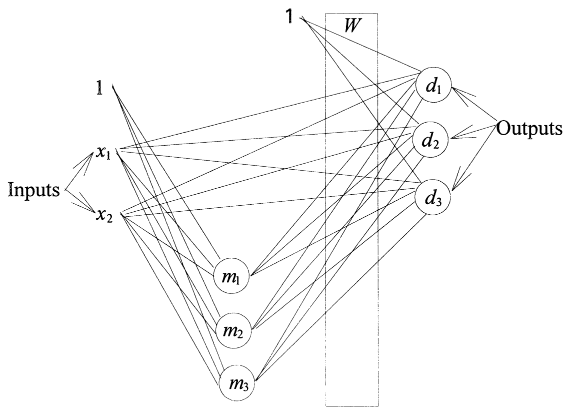
Because $X^{(1)},\dots,X^{(i-1)},X^{(i+1)},\dots,X^{(p)}$ inputs are to produce the same output 0 using a perceptron, they are on the same side of a hyperplane. ($X^{(i)}$ input is to produce the output 1 using a perceptron, it is on the another side of the hyperplane.) Training sets whose sample patterns have the above property are called linearly separable. So, input X (in (3)) and output H (in (5)) have perceptron solutions. □

From Lemmas 1 and 2, we can conclude the following theorem.



$$W' = \begin{bmatrix} -0.96 & -1.56 & 1.36 \\ -0.34 & 1.09 & -2.13 \\ -1.93 & 1.36 & 0.77 \end{bmatrix}$$

Fig. 3. The generation of higher-order input terms.



$$W' = \begin{bmatrix} -0.59 & 0.12 & 0.02 & 0.84 & 1.86 & -0.82 \\ -0.18 & 2.06 & -0.23 & -0.63 & 1.69 & 1.31 \\ 0.52 & 0.34 & -0.87 & 0.18 & 0.05 & -0.17 \end{bmatrix}$$

Fig. 4. The final network for Example 1.

Theorem 1. *The mapping $X \rightarrow D$, where X and D are arbitrary matrices of dimensions $(p \times n)$ and $(p \times 1)$, respectively, can be implemented with one layer of perceptron nodes followed by a perceptron node.*

The design procedure is composed of two phases. Phase 1 is the generation of higher-order input terms (H), in which we apply the multi-output binary perceptron training algorithm to the network. (Training pairs are $\{X^{(i)}, H^{(i)}\}$.) The outputs of the higher-order input terms are denoted by $M = [m_1\ m_2\ \dots\ m_k]'$. In phase 2, we use the training pairs $\{U^{(i)}, D^{(i)}\}$, where $U^{(i)} = X^{(i)} \cup M$, and $i = 1,2,\dots,p$, to be learned by the multi-output binary perceptron training algorithm.

Example 1. There are a 2-input and a 3-output truth table as shown in Table 1.

First, we have to generate the higher-order input terms. From Table 1, $p = 4$, $n = 2$, $X^{(1)} = [1\,0\,0]'$, $X^{(2)} = [1\,0\,1]'$, $X^{(3)} = [1\,1\,0]'$, $X^{(4)} = [1\,1\,1]'$ and

$$H = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Apply the multi-output binary perceptron training algorithm to this network with training pairs $\{X^{(i)}, H^{(i)}\}$. After being trained for 5 epochs, the network classifies correctly, and the result is shown as in Fig. 3. In phase 2, from Table 1, $D^{(1)} = [0\,0\,1]'$, $D^{(2)} = [1\,0\,0]'$, $D^{(3)} = [1\,1\,1]'$, $D^{(4)} = [0\,1\,0]'$, $X^{(1)} = [1\,0\,0\,0\,0\,0]'$, $X^{(2)} = [1\,0\,1\,1\,0\,0]'$, $X^{(3)} = [1\,1\,0\,0\,1\,0]'$ and $X^{(4)} = [1\,1\,1\,0\,0\,1]'$. Apply the multi-output binary perceptron training algorithm to this network with training pairs $\{U^{(i)}, D^{(i)}\}$. Trained for 4 epochs, the network classifies correctly, and the result is shown as in Fig. 4. Pick one set of data to verify the results, if the input vector is $[1\,0]'$ by any chance which coincide with the desired output values $[1\,1\,1]'$. From Fig. 3, we have

$$\begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = f(W'X) = f\left(\begin{bmatrix} -0.96 & -1.56 & 1.36 \\ -0.34 & 1.09 & -2.13 \\ -1.93 & 1.36 & 0.77 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

From Fig. 4, we have

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = f\left(\begin{bmatrix} -0.59 & 0.12 & 0.02 & 0.84 & 1.86 & -0.82 \\ -0.18 & 2.06 & -0.23 & -0.63 & 1.69 & 1.31 \\ 0.52 & 0.34 & -0.87 & 0.18 & 0.05 & -0.17 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ m_1 \\ m_2 \\ m_3 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

3. Adaptive dimension expansion

As mentioned in Section 2, we expand $p-1$ terms for p n-input train data as shown in (5). In the case of large number p will make the approach impractical. We hereby propose an adaptive expansion method to reduce the number of expanded terms. For example, if the system is linearly separable then we do not need to expand any more. Here we used an example of a network construction to introduce the concept of adaptive dimension expansion. The example neural network is constructed to realize the exclusive OR Boolean function. The desired input–output mapping is given in Table 2. The X and D matrices are then

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \qquad D = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

A solution to this equation exists if the rank of X is equal to the rank of $[X\ D]$, but A is of rank 3 and $[X\ D]$ is of rank 4, thus a single layer neural-network solution does not exist. Take first two rows of X, denoted by the form of $X(1{:}2,:)$, and the first two rows of D, denoted as $D(1{:}2)$. Now rank$(X(1;2,:))$ = rank$([X(1;2,:)\ D(1{:}2)])$ = 2, so a solution to this problem does exist. Similarly, we take first three rows of X, denoted as $X(1{:}3,:)$, and the first three rows of D, denoted as $D(1{:}3)$. Now rank$(X(1{:}3,:))$ = rank$([X(1{:}3,:)\ D(1{:}3)])$ = 3, so a solution to this problem does exist. We know that we cannot take $X(1{:}4,:)$ and $D(1{:}4)$. The fourth row of X is the place where we need to expand dimension by augmenting with a column vector. In this column vector, all are 0s except the fourth row is 1. Thus

$$U = \begin{bmatrix} & 0 \\ X & 0 \\ & 0 \\ & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

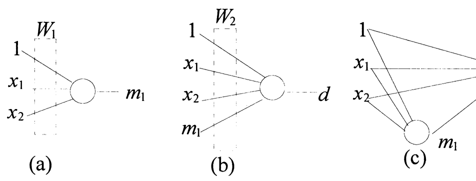
So, we have got

$$H = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

From Lemma 2, the data pairs (X,H) have a perceptron solution. The single-layer solution to the problem is

Table 2
The XOR function

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	0



$$W_1 = \begin{bmatrix} -2.3077 \\ 2.1780 \\ 1.8609 \end{bmatrix}, \quad W_2 = \begin{bmatrix} -0.1680 \\ 1.0539 \\ 0.1839 \\ -3.6922 \end{bmatrix}$$

Fig. 5. Components of the XOR neural network.

shown in Fig. 5(a). From Lemma 1, the rank of U is equal to the rank of $[U\ D]$. The single-layer solution to the problem is shown in Fig. 5(b). The network that realizes the XOR problem is shown in Fig. 5(c).

Fig. 6 gives the Matlab-code used to implement adaptive dimension expansion. The inputs to the algorithm include the training patterns X and D. The outputs of the algorithm include the U and H matrices. There are two main parts in the algorithm. The first one is to detect the redundancy of the input patterns. All redundancies should generate the same effect to the U and H matrices. The last one is to solve a succession of subsystems of linear equations. If the subsystem is linearly separable, then we do not need to expand anymore, otherwise we need to expand one column of H. The algorithm's solution is based on solving a succession of systems of linear equations. Some methods for solving the linear equation are recommended by (Moody and Antsaklis, 1996). These methods are guaranteed to converge in a number of steps equal to the order of the system, assuming that the problem is sufficiently well conditioned (Kincaid and Cheney, 1991).

Another simulation has been conducted in order to show the validity of the above design concept of the adaptive dimension expansion. The training patterns used in the simulation are 4 symbols, each being

```
function [U,H]=expand(X,y)
U=X;d=y;Ud=[U d];
[p,m]=size(U);
C1=zeros(p,1);j=1;
for i=2:p,
  ok=0;
  for k=1:i-1
    if X(i,:)==X(k,:)
      ok=1;
      U(i,:)=U(k,:);
      break;
    end
  end
  if ok==0
    if rank(U(1:i,:))~=rank(Ud(1:i,:))
      k=m+j;
      U(:,k)=C1;U(i,k)=1;
      Ud=[U d];
      I(j)=i; j=j+1;
    end
  end
end
E=U;
H=U(:,m+1:k);
```

Fig. 6. Algorithm 1 (Expansion).

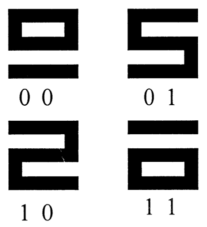


Fig. 7. Four non-linearly separable patterns.

represented by a 10×10 pixel matrix as shown in Fig. 7, and the corresponding targets are binary codes as shown under each pattern. The neural network used was the adaptive dimension expansion model with 100 inputs + 1 expansion bit and 2 (phase 1) + 3 (phase 2) outputs. After being trained 2 epochs, the network classifies correctly. For a non-linearly separable case as in Fig. 7, we cannot apply the typical perceptron network to solve it. But, it is easy to solve it by the adaptive dimension expansion method. It should be noted that these almost trivial examples are just means to show the detailed net configurations that come out of our design technique. The real strength of our technique can only be fully appreciated by testing mappings of extremely large dimensions. It is our goal to show such testing results in future study.

4. Conclusion

In this letter, we have proved that for the binary function if a desired output is expressed in sum of products, the desired output can be implemented with one layer of perceptron nodes followed by a perceptron node. This result is similar to the digital system, but we still have the benefits of neural networks.

References

Cybenko, G., 1988. Continuous valued neural networks with two hidden layers are sufficient. Technical Report, Dept. of Computer Science, Tufts University, Medford, MA.

Frean, M., 1990. The upstart algorithm: A method for constructing and training feed-forward neural networks. Neural Comput. 2, 198–209.

Gallant, S.I., 1986. Three constructive algorithms for network learning. In: Proc. 8th Ann. Conf. of the Cognitive Science Society, pp. 652–660.

Gray, D.L., Michel, A.N., 1992. A training algorithm for binary feedforward neural networks. IEEE Trans. Neural Networks 3 (2), 176–194.

Hornik, K., Stinchcombe, White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Networks 2, 359–366.

Kincaid, Cheney, W., 1991. Numerical Analysis. Brooks/Cole, Monterey, CA.

Mascioli, F.M., Martinelli, G., 1995. A constructive algorithm for binary neural networks: The oil-spot algorithm. IEEE Trans. Neural Networks 6 (3), 794–797.

Minsky, M., Papert, S., 1988. Perceptrons: An Introduction to Computational Geometry, Expanded edn. MIT Press, Cambridge, MA.

Moody, J.O., Antsaklis, P.J., 1996. The dependence identification neural network construction algorithm. IEEE Trans. Neural Networks 7 (1), 3–15.

Nadal, J.P., 1989. Study of a growth algorithm for a feedforward network. Internat. J. Neural Systems 1 (1), 55–59.

Pao, Y.H., 1989. Adaptive Pattern Recognition and Neural Networks. Addison-Wesley, Reading, MA.

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. 65, 386–408.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L. (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, Cambridge, MA, pp. 318–362.