Construction of Multi-layer Feedforward Binary Neural Network by a Genetic Algorithm

Chi Kin Chow and Tong Lee
Computer Vision and Image Processing Laboratory
Department of Electronic Engineering
The Chinese University of Hong Kong
{ckchow1, tlee}@ee.cuhk.edu.hk

Abstract *- A new approach is introduced to determine the topology of a feedforward binary neural network (BNN) automatically. The approach bases on a construction algorithm that constructs one layer of hidden nodes at a time until the problem is solved. And in each layer, the algorithm determines the necessary number of nodes through a growth process by finding the best hidden node that would help to partition the input training data set. This is done using a Genetic Algorithm. The proposed algorithm can determine the necessary number of hidden layers and number of hidden nodes at each layer automatically. Tests on a number of benchmark problems illustrated the effectiveness of the proposed technique, both in terms of network complexity and recognition accuracy, compared with a recent approach by geometrical learning.*

1. Introduction

Binary neural network is widely applied to many problems like pattern classification, error correcting and Boolean function mapping. There are many existing approaches to determine the weights of a neural network. The most popular and traditional approach is called back-propagation algorithm [1]. It is based on the error feedback and gradient descent method to update the weights of each hidden node iteratively. Usually, the topology of the multi-layer feedforward neural network has to be determined *a priori*. Therefore many attempted to determine the topology of a multi-layer neural network automatically. Growth or construction process is a popular approach. Cascade-correlation [2] is perhaps the most well-known and the early example of such approach. However, this approach can only be applied to certain type of topologies. For example, the cascade-correlation approach will determine a network with one hidden node at each hidden layer. A rather popular approach is to determine the one-hidden layer topology [3] because it has been proven that all problems can be solved with one-hidden layer architecture [4]. Recently, considerable attention has been given to train a multi-layer neural network with some global optimization method, such as Genetic Algorithm and Evolutionary Programming [5]. This approach still suffers from the following common drawbacks:

- The network architecture must be fixed *a priori*. i.e. the number of layers and the number of hidden nodes for each layer must be pre-determined by the user before the training process. Only the optimal weights are considered.
- If the topology is allowed to vary, the learning time will be, in general, very high. Consequently the number of weights can be considered is limited.

In this paper we address the problem of training a multi-layer feedforward network without specifying the topology by adopting the construction approach. However, the focus is limited to the construction of a feed-forward binary neural network with multi-layers structure because the training task is relatively easier [6-9]. We postulate that the function of each hidden layer in a feedforward binary neural network is a transformation mapping the training data from one data space to another. And we observe that the transformations implemented by each layer have the following characteristics:

- If two input data have been transformed to have the same output at a certain hidden layer, they cannot be distinguished at subsequent hidden layers.
- The number of distinguishable data after a transformation by the hidden node mapping cannot be more than the number of input data of the hidden layer.

Therefore, based on the above observations, we propose a new algorithm to construct a feedforward binary neural network. The algorithm aims at finding the transformations such that the number of input data can be reduced at each hidden layer while the data remains separable. This is done by maximizing the number of data with the same output after a hidden layer transformation. This has been formulated as an optimization problem and we propose to use GA to determine the fittest weights of each hidden node, one at a time. Then the hidden layers so obtained are connected to form a complete network. We evaluated the proposed algorithm against a recent approach by geometrical learning [6, 7] with benchmark problems.

0-7803-7278-6/02/$10.00 ©2002 IEEE

The organization of this paper is as follows. Section 2 introduces and defines the general structure and mechanism of BNN. In Section 3, a new algorithm to construct a BNN is discussed. Section 4 presents a method to determine the weights of a hidden node by a genetic algorithm, while Section 5 shows 3 sets of experimental results to illustrate the performance of the proposed algorithm. The paper is ended with the conclusion in Section 6.

2. Binary Neural Network (BNN)

Binary neural network is a special kind of artificial neural networks. During the training process, a set of input/output pattern pairs are given in order to determine the weights of the network, and every input pattern $[x_1, x_2, \ldots, x_n]$ and output pattern C pair is regarded as a training sample $\vec{x} = [x_1, x_1, \ldots, x_n \mid C]$. For the BNN, the possible values of x_i and C can either be 0 or 1. Figure 1 shows the general topology of a BNN.

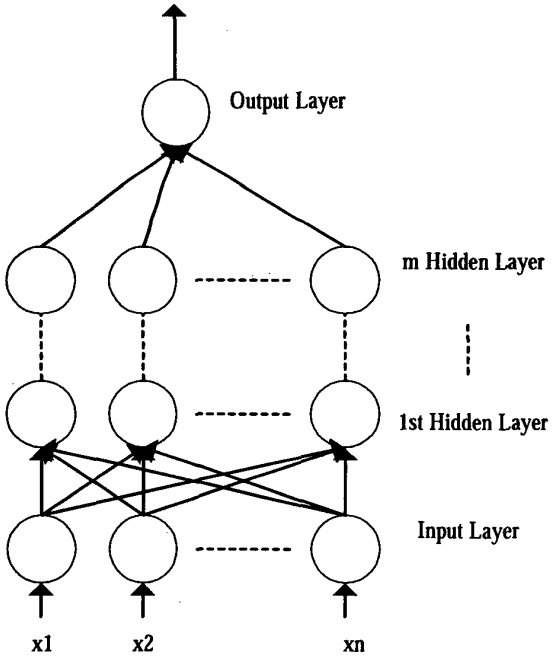


Figure 1 General topology of a BNN

For each hidden node with the weights $[\hat{n}, \rho]$, the input vector $\vec{x}$ and the output y can be related by $y = \mathrm{sgn}(\vec{x} \cdot \hat{n} + \rho)$ where

$$\mathrm{sgn}(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}.$$

The corresponding decision hyperplane h is defined as $h = \vec{x} \cdot \hat{n} + \rho$ and divides the input data space into partitions. Each partition can be classified as one of the following types:

1. Pure partition – the partition contains the training samples with the same output.
2. Mixed partition – the partition contains both training samples with output 0 and output 1.
3. Empty partition – no training sample can be found in the partition.

Since the training samples inside the same partition will be transformed to the same output pattern, hidden nodes with mixed partition will eventually introduce error to the network. Therefore, no mixed partition should be allowed.

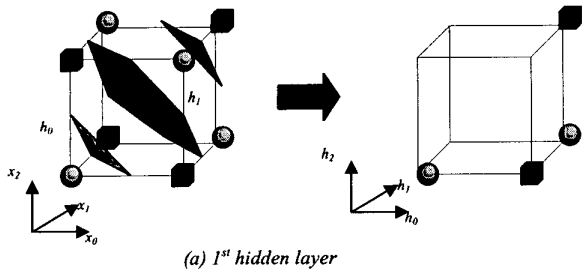
3. A new construction algorithm

The objective is to determine a hidden layer such that maximum number of training samples with the same output are grouped in the minimum number of partitions. In the ideal case, all training samples with the same output fall in the same partition, and because the output value is restricted to either 0 or 1 (2 possible outputs), one decision hyperplane is sufficient to classify the training samples. In practice, training samples with the same output will distribute on more than one partitions. Therefore, more than one hidden layers may be necessary. The number of partitions will be reduced after each layer and after passing through sufficient number of hidden layers, all training samples with the same output will fall in the same partition. Therefore, we can determine the required number of hidden layers for a given problem with this approach.

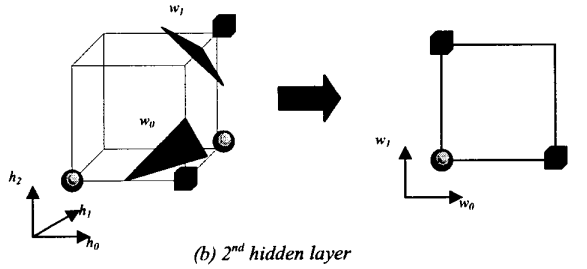
Figure 2 illustrates the proposed steps to determine the topology with a 3D parity binary pattern. At the first hidden layer, the training samples are distributed at the *x-domain* as shown in fig. 2a. 3 decision hyperplanes are used to separate the training samples into 4 groups. Then the 8 original training samples will be transformed into 4 training samples distributed over the *h-domain*. Similar to the first hidden layer, 2 decision hyperplanes are used to separate the training samples at the second hidden layer in the *h-domain* as shown in fig. 2b. Finally, only 1 decision hyperplane is necessary to separate the training samples at the output layer as shown in fig. 2c. Thus the topology of the corresponding binary neural network is completely determined as 4-2-1. This example merely illustrates the proposed steps in determining the topology of a multilayer network. In principle, only one decision hyperplane is necessary to separate the training samples in fig. 2b so only a two-layer topology is sufficient for this problem.

Therefore the essential step is to determine the decision hyperplanes of a hidden layer. We adopt the growth

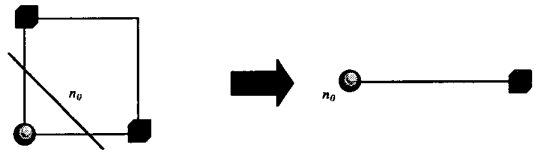
approach by determining one hyperplane at a time until all training samples are well partitioned. Figure 3 illustrates the growth process. Each hidden node attempts to partition as many training samples of the same class as possible. The complete algorithm is summarized below.



(a) 1st hidden layer



(b) 2nd hidden layer



(c) 3rd hidden layer

Figure 2 The 3D parity example for determining the BNN topology

Step 1: Decision hyperplane determination

We find a decision hyperplane $\bar{h}$, which can maximize the number of training samples inside the pure partition(s) created by the $\bar{h}$. We show in Section 4 that this is a non-linear optimization problem and we propose to use GA to solve it.

Step 2: Training sample elimination

After a decision hyperplane $\bar{h}$ is determined from step 1, the domain is divided into 2 partitions. All the training samples inside the pure partition will be eliminated as they already have been well partitioned by the $\bar{h}$.

Step 3: Layer completeness

As steps 1 and 2 are repeated t times, t decision hyperplanes are determined. And the current layer is completed if the partitions formed by those t decision hyperplanes are either pure or empty.

Step 4: Domain Transformation

When the current hidden layer is completed by a set of decision hyperplanes $\{\bar{h}_i\}$, *where* i = *1, 2, ...,* N_h *and* N_h *is the number of hyper planes in* $\{\bar{h}_i\}$, *the training sample vectors* $\bar{x}_i = [x_{i,1}, x_{i,2}, ..., x_{i,n} \mid C_i]$ *will be transformed into* $\bar{y}_i = [y_{i,1}, y_{i,2}, ..., y_{i,N_h} \mid C_i]$ *where* $y_{i,j} = \mathrm{sgn}(\bar{x}_i \cdot \hat{n}_j + \rho_j)$. *The dimension of the new domain is transformed from* n *to* N_h. *Some of the transformed training samples would be equivalent since they fall into the same partition. They would appear to be the same training pattern for the next hidden layer. For example, given the training sample set in Table 1a,* $\{\bar{x}_i\}$ *is transformed into* $\{\bar{y}_i\}$ *in Table 1b through the first hidden layer. We can observe that* $\bar{y}_1$, $\bar{y}_2$, *and* $\bar{y}_4$ *appear to be the same training data to the next hidden layer because they have the same set of input and output patterns. Likewise,* $\bar{y}_3$, $\bar{y}_5$, *and* $\bar{y}_6$ *are similar. Thus* $\bar{y}_2$, $\bar{y}_4$, $\bar{y}_5$, $\bar{y}_6$ *can be eliminated and the resultant transformed training samples for the next hidden layer become those listed in Table 1c.*

Step 5: Topology completeness

The topology of a binary neural network is completed if the training sample set can be classified by one decision hyperplane. The flow diagram of the proposed algorithm is shown in figure 4.

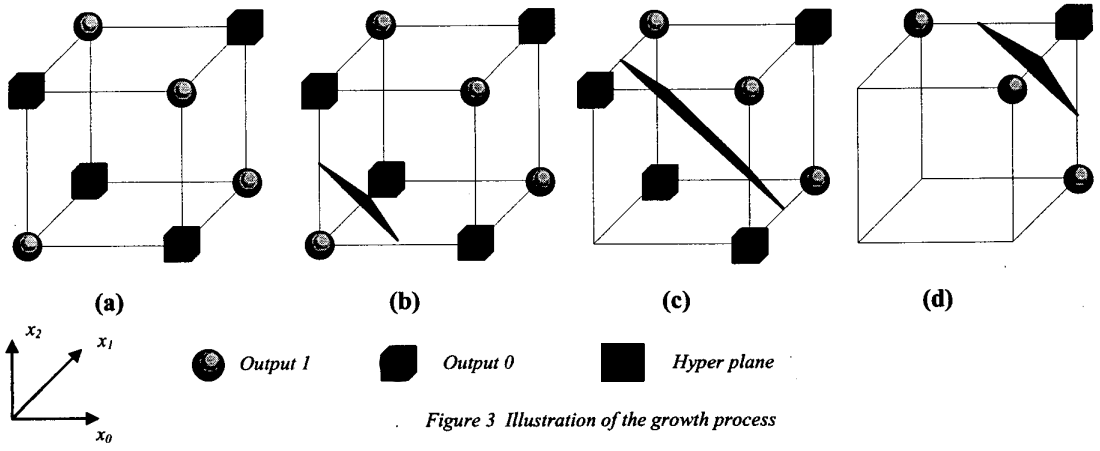


Figure 3 Illustration of the growth process

$\bar{x}_i$	C_i
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1

Table 1a

$\bar{y}_i$	C_i'
000	0
001	1
001	1
010	0
001	1
010	0
010	0
111	1

Table 1b

Resultant $\bar{y}_i$	Resultant C_i'
000	0
001	1
010	0
111	1

Table 1c

4. Genetic Algorithm

Genetic algorithms (GA) are search algorithms based on the mechanics of natural selection and natural genetics. A possible solution is represented as a chromosome in a string structure with each element representing one parameter in the solution. A collection of possible solutions (chromosomes) then forms a population, which produces another generation through a search process. The search process adopts "the fittest survives" rule after a structured yet randomized information exchange within the existing generation to yield a new generation. It has been successfully applied to solve many non-linear optimization problems. The success of a GA, however, depends on how to formulate the chromosomes and the fitness function, which are described in the following for determining the decision hyperplane required in Step 1 of Section 3 above.

A. Formation of genes and chromosome

Since the genetic algorithm is used to determine a decision hyperplane such that it can separate maximum number of training samples, the chromosome is used to represent the decision hyperplane and each gene of the chromosome is the parameter of the decision hyperplane. The general formation of an n-dimensional hyperplane is:

$$\sum_{i=1}^{n} x_i \times n_i + \rho$$

where $\hat{n} = [n_1, n_2, ..., n_n]$ is the normal direction of the hyperplane. Since $|\hat{n}| = 1$, we can use n – 1 parameters called *direction genes* ($\bar{\theta} = [\theta_1, \theta_1, ..., \theta_{n-1}]$) instead of n parameters to represent the direction. The normal direction can be determined by the equation:

$$n_i = \cos\phi_i \times \prod_{j=1}^{i-1} \sin\phi_j$$

$$\text{where } \bar{\phi} = [\bar{\theta}, 0] = [\phi_1, \phi_2, ..., \phi_{1n}]$$

Together with ρ, *the magnitude gene*, there are totally n genes (n – 1 direction genes and 1 magnitude gene) in a chromosome, i.e.

Chromosome = $[\theta_1\ \theta_2 \ldots \theta_{n-1}\ \rho]$ where $|\theta_i| \le \pi$ for $i = 1, 2, \ldots,$ n–1 and $0 \le \rho \le \sqrt{n}$.

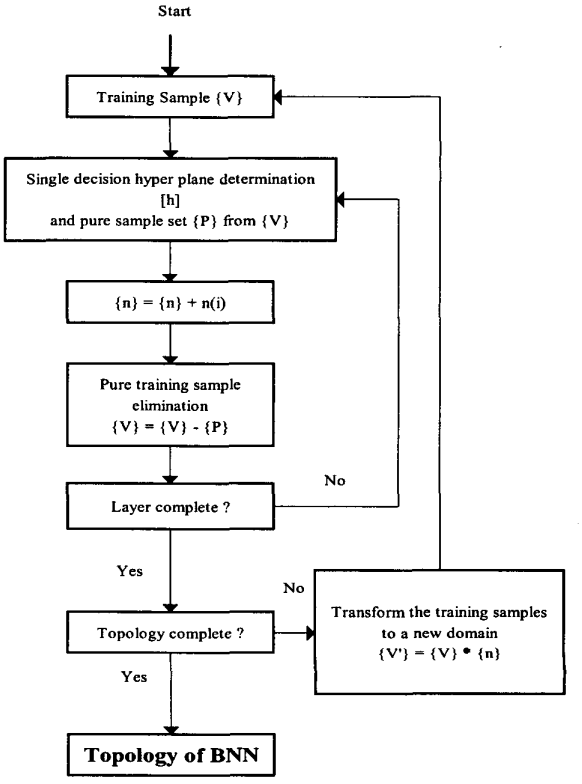


Figure 4 Flow diagram of proposed algorithm

B. Fitness Function

The fitness function should measure the performance of a neural network, which is, in general controlled by: 1) accuracy, 2) generalization, and 3) size of the network. In this paper, the fitness function used in this paper will just concern with the accuracy and the size of the network. Since our observations show that if the partitions produced a hidden layer is of the mixed type, error would be introduced. Moreover, if each partition contains more training samples, fewer partitions are needed and fewer hidden nodes are required. Therefore, we let the fitness function for a hidden node measure its partition purity and the GA should find the hidden node that generates maximum partition purity.

To define the partition purity of a hidden node, we consider how a hidden node partitions the input data. A hidden layer will divide the input space into 2 partitions P_+ and P_- with the following 4 possible combinations (assuming no empty partition):

- both P_+ and P_- are mixed
- P_+ is pure and P_- is mixed
- P_+ is mixed and P_- is pure
- both P_+ and P_- are pure.

Then we let the partition purity of a hidden node be $\boldsymbol{F} = f_+ + f_-$ where f_s defines the purity of the partition P_s for $s = -$ or $+$:

$$f_s = 0 \quad \text{if } P_s \text{ is mixed}$$
$$= \text{Size}(P_s) \quad \text{otherwise.}$$

For example, if the hyperplane of a hidden node divides the training data into two non-overlapping subsets, the fitness is 0 if both subsets consist of training data with two output classes. If one of the subsets consists of training data of the same class, then the fitness equals to the size of that subset. If both subsets happen to consist training data of the same class, the fitness equals to the total number of training data.

C. Reproduction

i. Cross-over

A multi-point cross-over operator has been adopted here. Given two chromosomes $h_i = [\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,n-1}, \rho_i]$ and $h_j = [\theta_{j,1}, \theta_{j,2}, \ldots, \theta_{j,n-1}, \rho_j]$, we first randomly select the number of genes to be swapped N_{swap} where $1 \le N_{swap} \le n$ as there are n genes in a chromosome. For each swapping, we select N_{swap} genes and then inter-change their values. All genes have the equal probability to be selected. Two new chromosomes are then generated.

ii. Mutation

After cross-over, each gene will then further go through the mutation operation. Each real-valued gene in a chromosome will be accumulated with a small value. The actual value to be accumulated is generated randomly within the range $[-\partial Gene_i, +\partial Gene_i]$. The range limits, $\partial Gene_i$ have been set to $\pi/4 = 0.7853$ and to $1/\sqrt{n}$ for the direction genes, θ and the magnitude gene, ρ respectively.

D. Selection

After the reproduction process, the N chromosomes with best fitness values will be selected to form the next generation.

5. Experimental Results

Experiment 1: N-bit parity functions

In this experiment, we apply the proposed algorithm described in Section 3 to classify n-bit parity problems. The parity functions are difficult Boolean functions to learn via NNs. The difficulty increases nonlinearly with n. Therefore, the parity functions are often used as benchmarks to test the capability of NNs. The number of bits in the parity function is varied from 2 to 9, which is sufficient to conclude the performance of the proposed algorithm. By observing the results listed in Table 2, we find that the successful rate is higher than 95% even though the problem is a high dimensional function. Moreover, the time to determine the corresponding topology is just varied from 0.5 to 10 sec.

Experiment 2: Two spiral function

The spiral function is another popular benchmark problem. Originally, it is a continuous function. However, since the BNN restricts the values of input pattern to be either 0 or 1, we first quantise the continuous 2D pattern to a 16 × 16 uniform grid pattern as shown in Fig. 5a. The input pattern now can be described by a discrete function *class* = $f(x, y)$ where x and y is an integer between 0 to 15. We convert x and y to binary codes and hence, x and y will form a binary input string pattern. Therefore the example has 128 training samples. Figures 5(b) and 5(c) illustrate the transformed patterns at the first and the second hidden layers where the pixels with same intensity belong to the same partition. Fig. 5(d) shows the output pattern of the BNN, which is the same as the target pattern. By comparing the transformed patterns in each hidden layer shown in fig. 5, we find that the training samples of the same class group together through each layer to form the target pattern in the output. In order to compare the performance between the proposed algorithm and the modified geometrical learning reported in [6], we apply the proposed algorithm on the same double spiral function used in [6]. The corresponding hidden nodes required are only 15 comparing to 19 in [6]. The topology determined using the proposed algorithm is 8 - 7 - 1 while only one hidden layer is adopted in [6]. Moreover in our formulation, we did not generate additional virtual training data as in [6]. The virtual training data are additional data generated from the original set of training data without altering the function mapping. This experiment was done with 10 trails. 8 out of 10 trails converged to the same topology while the remaining two converged to the solutions with more hidden nodes.

Experiment 3: A 7-bit function

We apply the proposed algorithm to determine a 7-bit Boolean function in [6]. There are 36 input patterns with output 0 and 36 input patterns with output 1 while others are don't care. The proposed algorithm requires the same number of hidden nodes as [6] without generating the virtual training data. The experiment was repeated 10 times and all 10 trials converged to the optimal topology.

6. Conclusion

We have proposed an efficient algorithm to determine the topology of a binary neural network automatically by maximizing the purity of data distribution at each hidden

layer. A GA is proposed for handling the optimization process. The experimental results show that the proposed algorithm is more effective than the geometrical learning approach in constructing a binary neural network without adding virtual training data. Extension of the current work on multilayer perceptron is under investigation by incorporating the generalization in the fitness function.

7. References

[1] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing – Explorations in the Microstructure of Cognition*, Vol. 1 and 2, Cambridge: MIT Press, 1986.

[2] S.E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in Advances in Neural Information Processing Systems II, D.S. Touretzky (Ed.), San Mateo, CA: Morgan Kaufmann, 1990, pp. 524-532.

[3] F. L. Chung and T. Lee, "A network growth approach to the design of feedforward neural networks", *IEE Proceedings on Control Theory and Applications*, vol. 142, no. 5, Sept. 1995, pp.486-492.

[4] K. Hornik, "Multilayer feedforward networks are univerisal approximator," Neural Networks, vol. 2, no. 5, 1989, pp. 359-366.

[5] Mu-Song Chen and Fong Hang Liao, "Neural networks training using genetic algorithms", in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, 1998, vol. 3, pp. 2436-2441.

[6] M. Shimada and T. Saito, "A simple learning of binary neural networks with virtual teacher signals," in *Proc. International Joint Conference on Neural Networks*, 2001, Vol. 3 , pp. 2042-2047.

[7] J.H.Kin and S.K.Park, "the geometrical learning of binary neural networks", *IEEE Transactions Neural Networks*, vol. 6, no. 1, Jan. 1995, pp. 237-247.

[8] D.L. Gray and A.N. Michel, "A training algorithm for binary feedforward neural networks", *IEEE Transactions on Neural Networks*, vol. 3, no. 2, March 1992, pp. 176-194.

[9] M. Muselli, "On sequential construction of binary neural networks ", *IEEE Transactions on Neural Networks*, vol. 6, no. 3, May 1995, pp. 678-690.

N	*Expected Topology*	*Determined Topology*	*# of successful trials out of 30 trials*	*Processing time (sec.)*
2	[2, 1]	[2, 1]	30	0.5
3	[3, 2, 1]	[3, 2, 1]	30	0.8
4	[4, 2, 1]	[4, 2, 1]	30	1.2
5	[5, 2, 1]	[5, 2, 1]	30	2.5
6	[6, 2, 1]	[6, 2, 1]	29	4.7
7	[7, 2, 1]	[7, 2, 1]	29	6.8
8	[8, 2, 1]	[8, 2, 1]	29	8.5
9	[9, 2, 1]	[9, 2, 1]	28	9.7

Table 2 Results of N - parity experiments

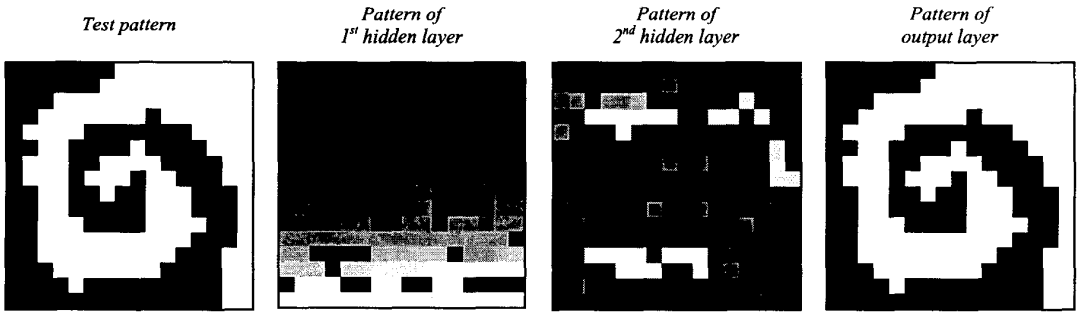


Figure 5 Result of the two-spiral experiment