2015 International Conference on Computational Science and Computational Intelligence

A New Scheme for Implementing S-box Based on Neural Network

Xia Zhang,Fangyue Chen,Bo Chen,and Zhongwei Cao
School of Science
Hangzhou Dianzi University, Hangzhou, Zhejiang 310018, P. R. China
Email: heiyedejingling@163.com / fychen@hdu.edu.cn

***Abstract*—S-box (Substitution box) is one of the most important components in the block cipher. As the high nonlinearity of neural network (or artificial neural network, ANN) is in high accordance with the properties of cipher, the application of neural network in cryptography becomes a significant orientation. In this paper, we present a new scheme for implementing S-box used in ciphers basing on neural network. Differing from the previous network models, the proposed network, which can be used to implement any Boolean function in S-box, consists of multiple neural network perceptrons, and each perceptron only has a low number of input variables (4-bits input). By DNA-like learning algorithm, it is very convenient to train the weight and threshold values of the network.**

***Keywords*-Cellular Neural Network(CNN); S-box; SLP; MLP; Boolean Function(BF)**

I. Introduction

The S-box (substitution box), which is considered as the core of the block ciphers, is the only nonlinear element assuring the confusion property of the conventional block ciphers such as Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). The strength of the encryption depends on the ability of S-box in distorting the data. Hence, the processes of discovering new and powerful S-boxes are of great interest in the field of cryptography. The study of the S-box design accelerates the development of cryptography.

Recently, the application of neural network (or artificial neural network, ANN) in cryptography and cryptographic analysis has become increasingly interesting. Recent works have examined the use of neural networks to different components of cryptography systems [1-3].

In the field of cryptography, an S-box is a component of symmetric key algorithm which performs substitution. In block ciphers, it is typically used to obscure the relationship between the key and the ciphertext. In many cases, the S-boxes are carefully chosen to resist cryptanalysis. In general, an S-box takes some number of input bits, denoted by $\boldsymbol{m}$, and transforms them into some number of output bits, denoted by $\boldsymbol{n}$, and has some cryptographic properties, such as non-linearity, completeness, strict avalanche, output bits independency criteria and so on[4-7].

In this paper, we present a new scheme for implementing the S-box used in ciphers on neural network. The proposed network consists of some multiple-layer neural network perceptrons, and each only has an input of 4-bits. Therefore, it is very convenient to train the weight and threshold values of the network through DNA-like learning algorithm.

The rest of this paper is organized as follows. Section II gives some preliminaries on neural networks and S-box theory. In Section III, we describe the new S-box design scheme based on neural networks. Section IV gives some concluding remarks.

II. Overview of Neural Networks and S-Box

A. SLP and MLP of binary neural networks

Artificial neural networks have been motivated from their inception by the recognition that the brain computes in an entirely different way from the conventional digital computer. The brain contains billions of neurons with massive interconnections. Similarly, artificial neural networks are massively parallel-distributed processors that are made up of artificial neurons with interconnections. These are non-linear dynamic machines which expand the expression of input data as a linear combination of inputs and synapses and then perform a non-linear transformation to compute the output. There are some neural networks' properties which make them suitable to use in cryptology. These neural networks' properties include one-way property, parallel implementation, non-linear computations, confusion property, diffusion property, and so on[8].

An (m, n) S-box in the block cipher is a map $S : \{0,1\}^m \to \{0,1\}^n$, i.e., the input and output values of an S-box which consist of n Boolean functions are all binary, so the implementation of S-box only needs binary neural networks (BNN) which is a branch of neural network.

It is well known that single-layer perceptron (SLP) of binary neural networks can only be used to implement the class of linearly separable Boolean functions (LSBF), but the majority of Boolean functions are not linearly separable, so multi-layer perceptron (MLP) must be used to implement the class of non-linearly separable Boolean functions (non-LSBF). Of course, MLP can contain one or more hidden layers, but it was proved that MLP with one hidden layer can be used to perform any non-LSBF by using suitable learning algorithms [9, 10]. The expressions of SLP and MLP with one hidden layer are respectively: **(a)** SLP with

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.9

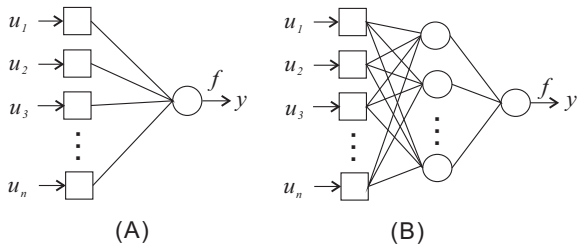

Figure 1. (A) Single-layer perceptron (SLP), (B) Multi-layer perceptron (MLP) with one hidden layer.

a hard-limitation activation function f:

$$y = f(\sum_{i=1}^{n} \omega_i u_i - \theta), \tag{1}$$

where f is the first-order jump function defined by

$$f(x) = sign(x) = \begin{cases} 1 & if \quad x > 0 \\ 0 & if \quad x \leq 0, \end{cases} \tag{2}$$

ω_i $(i = 1, 2, \cdots, n)$ are weight values and θ is threshold value, $u_i \in \{0, 1\}$ $(i = 1, 2, \cdots, n)$ are inputs, and y is the output.

(b) MLP consists of m SLPs connecting the input layer and the hidden layer as well as one SLP connecting the hidden layer and the output layer, as follows:

$$\begin{cases} y_j = f(\sum_{i=1}^{n} \omega_{ij} u_i - \theta_j) \ (j = 1, 2, \cdots, m) \\ y = f(\sum_{j=1}^{m} \bar{\omega_i} y_j - \bar{\theta}), \end{cases} \tag{3}$$

where y_j is the output from the $j-th$ neuron in the hidden layer, w_{ij} and θ_j are the connection weight-threshold values between the i-th input and j-th neuron in the hidden layer, and $\bar{w_j}$ and $\bar{\theta}$ are the weight-threshold values of the neuron of the output layer.

The topological structure of both SLP and MLP is shown in Figure 1.

As mentioned above, the MLP with one hidden layer can perform any Boolean function, but no matter which kind of learning algorithm one use, training the weight-threshold values of MLP is very difficult when the number of its inputs is large. However, we will use a method of reducing the dimension to tackle with this difficulty.

First, we give an important theorem as follows:

Theorem 1: Let $y^{(1)} = f^{(1)}(x_1, x_2, \cdots, x_{n-1})$ and $y^{(2)} = f^{(2)}(x_1, x_2, \cdots, x_{n-1})$ are two $(n-1)$-bit Boolean functions, $f^{(i)} : \{0, 1\}^{n-1} \to \{0, 1\}$ $(i = 1, 2)$, then

(a) $y = f(x) = [(1 \otimes x_0) \cdot y^{(1)}] \otimes x_0 \cdot y^{(2)}$ is an n-bit Boolean function, where $x = (x_0, x_1, x_2, \cdots, x_{n-1})$, "$\otimes$" and "$\cdot$" are the "$XOR$" and "$AND$" logical operations respectively.

(b) The expression of the MLP implementing the function

Table I
TRUTH TABLE OF THE BOOLEAN FUNCTION $y = f(x) = [(1 \otimes x_0) \cdot y^{(1)}] \otimes x_0 \cdot y^{(2)}$

x_0	$y^{(1)}$	$y^{(2)}$	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table II
WEIGHT $w_1, w_2,\ w_3$ AND THRESHOLD VALUE θ OF THREE LSBFS IN 172P.

Output bits	w_1	w_2	w_3	θ
00000001	4	6	2	11
00000011	4	6	2	9
00110111	4	6	2	5

$y = f(x) = [(1 \otimes x_0) \cdot y^{(1)}] \otimes x_0 \cdot y^{(2)}$ is

$$\begin{cases} u_1 = sign(4x_0 + 6y^{(1)} + 2y^{(2)} - 11) \\ u_2 = sign(4x_0 + 6y^{(1)} + 2y^{(2)} - 9) \\ u_3 = sign(4x_0 + 6y^{(1)} + 2y^{(2)} - 5) \\ y = sign(4u_1 - 2u_2 + 2u_3 - 3), \end{cases} \tag{4}$$

and the truth table of the Boolean function is shown in Table I. Since the decimal code of the output bits (00110101) of the Boolean function in the Table is 172, so the MLP is named as No.172 perceptron, and 172P in abbreviation .

(c) The networks implementing the Boolean function $y = f(x) = [(1 \otimes x_0) \cdot y^{(1)}] \otimes x_0 \cdot y^{(2)}$ consist of two $(n-1)$-bit perceptrons and a 172P.

Proof: (a) is obvious, and if the single input x_0 and the functions $y^{(1)}$ and $y^{(2)}$ are regarded as 3 input variables of a Boolean function, and y is regarded as the output of the function, so Table I in (b) was established. The output sequence of the function is (00110101). It is easy to test that the function is not linearly separable, but it can be decomposed as the logical XOR operations of three linearly separable Boolean functions (LSBFs), i.e., $00110101 = 00000001 \otimes 00000011 \otimes 00110111$. Further, based on DNA-like learning algorithm [9, 10], the weight-threshold values of the neurons implementing the three LSBFs and the ones of the neuron from the hidden layer to output layer are easy to be trained. The weights $w_1, w_2,\ w_3$ and threshold value θ of the three LSBFs are shown in Table II, and the weights values and the threshold value of the neuron from the hidden layer to output layer are respectively $(\bar{w}_1, \bar{w}_2,\ \bar{w}_3) = (4, -2, 2)$ and $\bar{\theta} = 3$. Thus, the expression of $172P$ is (4). Further, from above (a) and (b), the structure of the network implementing the function $y = f(x) = [(1 \otimes x_0) \cdot y^{(1)}] \otimes x_0 \cdot y^{(2)}$ is given, which consists of perceptron P1 and P2 combining with a 172P. It

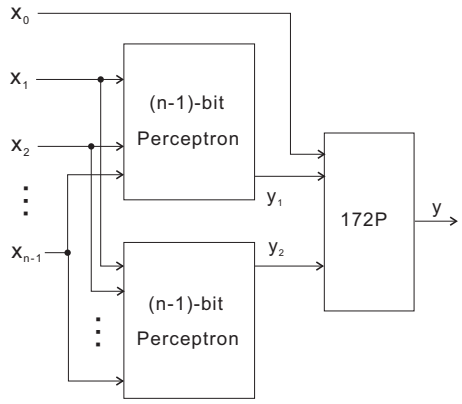

Figure 2. Networks implementing the Boolean function $y = f(x)$.

is shown in Figure 2.

■

Example1: For a 5-bit Boolean function $y = f(x)$, $x = (x_1, x_2, \cdots, x_5)$, its output-bits is $Y = (010100010101\ 11111101001011010101)$. Let $Y = (Y_1, Y_2)$, where $Y_1 = (0101000101011111)$, $Y_2 = (1101001011010101)$. Y_1 and Y_2 are respectively the outputs of two 4-bit Boolean functions. Based on the above results, one only needs to realize the two Boolean functions, i.e., one just looks for two perceptrons, P_1 and P_2, which can perform the two functions respectively. P_1 and P_2 combining with a $172P$ form a network which can perform the 5-bit Boolean function. In fact, P_1 and P_2 are two MLPs, and their weight-threshold values can easily be given by DNA-like learning algorithm. They are shown in Table III.

Remark1: A notable advantage of DNA-like leaning algorithm is that all the weight values of the neurons implementing all LSBFs, decomposed from an non-LSBF, are the same except the threshold values.

B. S-Box

The security of data relies on the substitution process. Substitution is an nonlinear transformation which performs confusion of bits. In modern encryption algorithm, an nonlinear transformation is essential and is proved to be a strong cryptographic primitive against linear and differential cryptanalysis. Especially important, the strength of product ciphers mainly depends on the properly designed S-boxes [6].

As mentioned above, an (m, n) S-box is a map $S: \{0,1\}^m \to \{0,1\}^n$. It comprises of n m-bit Boolean functions: $f_i(x_1, x_2, \cdots, x_m)$ $(i = 1, 2, \cdots, n)$. There are several criteria for designing S-box which are believed to be essential in the design of cryptographic algorithms. If an S-box does not satisfy one of the criteria, the cryptographic design based on the S-box may be cryptographically weak or easy to be attacked. The criteria which are considered essential for designing an S-box include balance property, completeness criterion, strict avalanche criterion (SAC), non-linearity criterion, bit independence criterion and so on. Of course, the strict avalanche criterion(SAC) is the most important in all of the properties. A change in one bit of input bits of an S-box should produce a change in half of the output bits of the S-box, which makes it harder to perform an analysis on the cipher text when trying to come up with an attack. A cryptographic function which satisfies the condition is said to be satisfied with strict avalanche criteria [1, 6].

III. A New S-box Design Scheme Based on Neural Network

A. An existing S-box and its Boolean functions

In [1], an $(8, 8)$ S-box was designed based on neural network, and the output values of the S-box are shown in Table IV. But, the S-box was performed by an MLP with two hidden layers, and the number of the neurons in the network is relatively large. Thus, it is difficult to train the wight-threshold values of the networks. We present a new scheme for implementing the S-box.

Let $x = (x_1, x_2, \cdots, x_8) \in \{0,1\}^8$, thus, the eight 8-bit Boolean functions contained in the S-box are $f_i(x)$ $(i = 1, 2, \cdots, 8)$. Due to space limitation, we just take $f_1(x)$ as an example to show the working pattern of our scheme.

The number of the output bits of each Boolean function is 256. For example, $f_1(x)$ is

$f_1(x) = 00001001000011101110100111111101111000110111010001010100111010010000010011010101010110011000011111000101000011010111011110011110010100110000000101001101101001100000001110111011001101001100101000011111000111001000100001110111100111101110011111011000100101 0.$

Indeed, it is very difficult to realize these functions via neural network using existing learning algorithm. However, we will perform them by the decomposition method in Theorem 1 in Section 2 and DNA-like learning algorithm.

B. A new framework implementing Boolean function

The 256 output bits of any 8-bit Boolean functions $f_i(x)$ $(i = 1, 2, \cdots, 8)$ in an $(8, 8)$ S-box can be decomposed into 2 parts, each of which contains 128 bits and can be considered as the output bits of a 7-bit Boolean function. Then, each 7-bit Boolean function can be decomposed into two 6-bit Boolean functions. In this way, until the function is decomposed into 16 4-bit Boolean functions.

Based on the results obtained in Theorem 1, an 8-bit Boolean function can be implemented using a network consisted of two perceptrons which perform respectively two 7-bit Boolean functions combining with a special perceptron 172P that can perform a 3-bit Boolean function. Similarly, a 7-bit Boolean function can be implemented by a network consisted of two perceptrons performing two 6-bit Boolean functions respectively in combination with 172P. In this

Table III
WEIGHT-THRESHOLD VALUES OF P_1 AND P_2.

Perceptron	m (number of hidden layer neurons)	Weight-threshold values of hidden layer neurons $\omega_1, \omega_2, \omega_3, \omega_4, \theta_1, \theta_2, \cdots, \theta_m$	Weight-threshold values of output layer neuron $\bar{\omega}_1, \bar{\omega}_2, \cdots, \bar{\omega}_m, \bar{\theta}$
P_1	3	$\omega_1 = 10, \omega_2 = 8, \omega_3 = 2, \omega_4 = 4$ $\theta_1 = 13, \theta_2 = 7, \theta_3 = 3$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2,$ $\bar{\theta} = 1$
P_2	4	$\omega_1 = -10, \omega_2 = 8, \omega_3 = -2, \omega_4 = 4$ $\theta_1 = 7, \theta_2 = -1, \theta_3 = -5, \theta_4 = -11$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2,$ $\bar{\omega}_4 = -2, \bar{\theta} = 1$

Table IV
OUTPUT VALUES OF THE S-BOX PROPOSED IN [1].

99	124	184	18	216	77	243	196	164	31	16	216	254	215	251	175
11	195	121	131	230	103	194	117	199	214	228	69	80	196	34	224
232	189	125	74	11	2	245	116	195	17	158	225	178	114	207	159
226	185	33	209	23	102	153	213	99	18	213	36	30	41	14	187
135	38	169	85	153	55	84	246	0	26	205	220	85	164	125	194
157	12	243	127	54	82	18	8	209	131	101	166	150	195	91	92
144	234	127	67	143	38	142	209	94	5	216	191	252	196	195	184
178	22	248	70	111	144	26	27	252	158	242	65	172	99	249	51
186	89	223	223	134	238	150	12	67	196	207	151	206	197	101	169
208	220	179	224	247	64	80	244	168	124	92	134	10	161	36	91
174	242	99	74	110	242	187	101	139	220	67	225	29	230	34	19
100	150	99	118	125	168	132	162	81	251	146	117	113	74	99	37
110	226	37	62	16	53	210	220	23	182	213	199	207	176	20	143
220	135	143	232	118	151	5	23	93	215	246	179	60	173	44	99
225	135	99	250	244	183	8	126	93	11	95	159	206	85	141	196
77	99	90	33	3	67	47	212	186	66	252	195	193	146	109	3

way, until a 5-bit Boolean function can be implemented by a network of two 4-bit perceptrons combining with 172P. Finally we take 4-bit Boolean function as the target to be implemented on a 4-bit perceptron, for a 4-bit Boolean function is very easy to be realized via SLP or MLP of the neural networks.

Thus, the steps of implementing a given 8-bit Boolean function $y = f(x)$ via neural network are as follows:

step 1: Let $Y = (v_1, v_2, \cdots, v_{256})$ be the output bits of Boolean function $y = f(x)$, and divide it into 16 parts: $Y = (Y^{(1)}, Y^{(2)}, Y^{(3)}, Y^{(4)}, Y^{(5)}, Y^{(6)}, Y^{(7)}, Y^{(8)}, Y^{(9)}, Y^{(10)}, Y^{(11)}, Y^{(12)}, Y^{(13)}, Y^{(14)}, Y^{(15)}, Y^{(16)}) = (Y_{1111}, Y_{1112}, Y_{1121}, Y_{1122}, Y_{1211}, Y_{1212}, Y_{1221}, Y_{1222}, Y_{2111}, Y_{2112}, Y_{2121}, Y_{2122}, Y_{2211}, Y_{2212}, Y_{2221}, Y_{2222})$, where

$$Y^{(i)} = (v_{2^4\cdot(i-1)+1}, v_{2^4\cdot(i-1)+2}, \cdots, v_{2^4\cdot i}), \quad (i = 1, 2, \cdots, 16). \tag{5}$$

For example, $Y^{(1)} = Y_{1111} = (v_1, v_2, \cdots, v_{16})$, $Y^{(2)} = Y_{1112} = (v_{17}, v_{18}, \cdots, v_{32})$ et.al.

step 2: Let

$$\begin{aligned} &Y_{111} = (Y_{1111}, Y_{1112}), Y_{112} = (Y_{1121}, Y_{1122}),\\ &Y_{121} = (Y_{1211}, Y_{1212}), Y_{122} = (Y_{1221}, Y_{1222}),\\ &Y_{211} = (Y_{2111}, Y_{2112}), Y_{212} = (Y_{2121}, Y_{2122}),\\ &Y_{221} = (Y_{2211}, Y_{2212}), Y_{222} = (Y_{2221}, Y_{2222}). \end{aligned} \tag{6}$$

Moreover

$$\begin{aligned} &Y_{11} = (Y_{111}, Y_{112}), Y_{12} = (Y_{121}, Y_{122}),\\ &Y_{21} = (Y_{211}, Y_{212}), Y_{22} = (Y_{221}, Y_{222}). \end{aligned} \tag{7}$$

Also

$$Y_1 = (Y_{11}, Y_{12}), Y_2 = (Y_{21}, Y_{22}). \tag{8}$$

Finally $Y = (Y_1, Y_2)$.

step 3: Each $Y^{(i)},\ i = (1, 2, \cdots, 16)$ is considered as the output bits of a 4-bit Boolean function, and they would be implemented via 16 4-bit perceptrons (SLPs or MLPs) using DNA-like learning algorithm as the implementing method in example 1 above. These perceptrons are named as $P_1, P_2, \cdots, P_{16}$ respectively.

step 4: Construct 8 networks using the following method: the output $Y^{(1)}$ of P_1, the output $Y^{(2)}$ of P_2, and the comment x_4 of input variables $x = (x_1, x_2, \cdots, x_8)$ are considered as the inputs of $172P$, thus getting a network which can perform the 5-bit Boolean function $Y_{111} = (v_1, v_2, \cdots, v_{32})$. Similarly, P_3 and P_4 combining with $172P$ can perform $Y_{112} = (v_{33}, v_{34}, \cdots, v_{64})$, P_5 and P_6 combining with $172P$ can perform $Y_{121} = (v_{65}, v_{66}, \cdots, v_{96})$, P_7 and P_8 combining with $172P$ can perform $Y_{122} = (v_{97}, v_{98}, \cdots, v_{128})$, P_9 and P_{10} combining with $172P$ can perform $Y_{211} = (v_{129}, v_{130}, \cdots, v_{160})$, P_{11} and P_{12} combining with $172P$ can perform $Y_{212} = (v_{161}, v_{162}, \cdots, v_{192})$, P_{13} and P_{14} combining with

Table V
WEIGHT-THRESHOLD VALUES OF A PART OF PERCEPTRONS IN THE NETWORK IMPLEMENTING BOOLEAN FUNCTION $f_1(x)$ IN THE $(8,8)$ S-BOX IN [1]

Perceptron	m (number of hidden layer neurons)	Weight-threshold values of hidden layer neurons $\omega_1, \omega_2, \omega_3, \omega_4, \theta_i \ (i = 1, 2, \cdots, m)$	Weight-threshold values of output layer neuron $\bar{\omega}_1, \bar{\omega}_2, \cdots, \bar{\omega}_m, \bar{\theta}$
P_1	3	$\omega_1 = -2, \omega_2 = 8, \omega_3 = 2, \omega_4 = 2$ $\theta_1 = 11, \theta_2 = 9, \theta_3 = 5$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2$ $\bar{\theta} = 1$
P_2	2	$\omega_1 = -2, \omega_2 = 6, \omega_3 = 6, \omega_4 = 4$ $\theta_1 = 13, \theta_2 = -9$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2$ $\bar{\theta} = 1$
P_3	4	$\omega_1 = 2, \omega_2 = 6, \omega_3 = -2, \omega_4 = -4$ $\theta_1 = 5, \theta_2 = 3, \theta_3 = 1, \theta_4 = -5$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2, \bar{\omega}_4 = -2$ $\bar{\theta} = 1$
P_4	4	$\omega_1 = -4, \omega_2 = -2, \omega_3 = -10, \omega_4 = -6$ $\theta_1 = -5, \theta_2 = -9, \theta_3 = -13, \theta_4 = -21$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2, \bar{\omega}_4 = -2$ $\bar{\theta} = 1$
P_5	3	$\omega_1 = 2, \omega_2 = -4, \omega_3 = 2, \omega_4 = 6$ $\theta_1 = 9, \theta_2 = 7, \theta_3 = 3,$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2$ $\bar{\theta} = 1$
P_6	3	$\omega_1 = -4, \omega_2 = -4, \omega_3 = 8, \omega_4 = 6$ $\theta_1 = 13, \theta_2 = 7, \theta_3 = 1$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2$ $\bar{\theta} = 1$
P_7	4	$\omega_1 = 4, \omega_2 = 10, \omega_3 = 6, \omega_4 = 2$ $\theta_1 = 17, \theta_2 = 9, \theta_3 = 5, \theta_4 = 3$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2, \bar{\omega}_4 = -2$ $\bar{\theta} = 1$
P_8	2	$\omega_1 = 4, \omega_2 = -2, \omega_3 = 2, \omega_4 = -6$ $\theta_1 = 3, \theta_2 = -1$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2$ $\bar{\theta} = 1$
P_9	2	$\omega_1 = 2, \omega_2 = 4, \omega_3 = 4, \omega_4 = -2$ $\theta_1 = 3, \theta_2 = -1$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2$ $\bar{\theta} = 1$
P_{10}	3	$\omega_1 = 4, \omega_2 = 8, \omega_3 = 6, \omega_4 = 10$ $\theta_1 = 23, \theta_2 = 17, \theta_3 = 9$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2$ $\bar{\theta} = 1$
P_{11}	1	$\omega_1 = 6, \omega_2 = -2, \omega_3 = -4, \omega_4 = -4$ $\theta_1 = -1$	
P_{12}	2	$\omega_1 = 6, \omega_2 = 2, \omega_3 = -4, \omega_4 = 2$ $\theta_1 = 5, \theta_2 = -1$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2$ $\bar{\theta} = 1$
P_{13}	2	$\omega_1 = -2, \omega_2 = -8, \omega_3 = 6, \omega_4 = 4$ $\theta_1 = 3, \theta_2 = -7$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2$ $\bar{\theta} = 1$
P_{14}	4	$\omega_1 = -8, \omega_2 = -2, \omega_3 = 4, \omega_4 = -6$ $\theta_1 = 1, \theta_2 = -9, \theta_1 = -13, \theta_1 = -15$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2, \bar{\omega}_4 = -2$ $\bar{\theta} = 1$
P_{15}	3	$\omega_1 = -4, \omega_2 = -2, \omega_3 = 4, \omega_4 = -6$ $\theta_1 = -1, \theta_2 = -3, \theta_3 = -9$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2$ $\bar{\theta} = 1$
P_{16}	3	$\omega_1 = -8, \omega_2 = 6, \omega_3 = -12, \omega_4 = -4$ $\theta_1 = -5, \theta_2 = -11, \theta_3 = -15$	$\bar{\omega}_1 = 2, \bar{\omega}_2 = -2, \bar{\omega}_3 = 2$ $\bar{\theta} = 1$

$172P$ can perform $Y_{221} = (v_{193}, v_{194}, \cdots, v_{224})$, P_{15} and P_{16} combining with $172P$ can perform $Y_{222} = (v_{225}, v_{226}, \cdots, v_{256})$. Then, in a similar way, we can construct 4 networks which can deal with 4 6-bit Boolean functions from the 8 networks obtained above and $172P$ combining with the comment x_5 of input variable x: $Y_{11} = (v_1, v_2, \cdots, v_{64})$, $Y_{12} = (v_{65}, v_{66}, \cdots, v_{128})$, $Y_{21} = (v_{129}, v_{130}, \cdots, v_{192})$, $Y_{22} = (v_{193}, v_{194}, \cdots, v_{256})$. Further, we can construct 2 networks which can deal with 2 7-bit Boolean functions $Y_1 = (v_1, v_2, \cdots, v_{128})$ and $Y_2 = (v_{129}, v_{130}, \cdots, v_{256})$. Finally, a network that can perform Y is got.

The framework of the network is shown in Figure 3.

Obviously, the key is realizing a 4-bit Boolean function to implementing an 8-bit Boolean function in an $(8,8)$ S-box.

C. The weight-threshold values of networks

Based on the network framework of implementing Boolean functions, the training of 16 perceptrons in the network performing the Boolean function in the S-box is

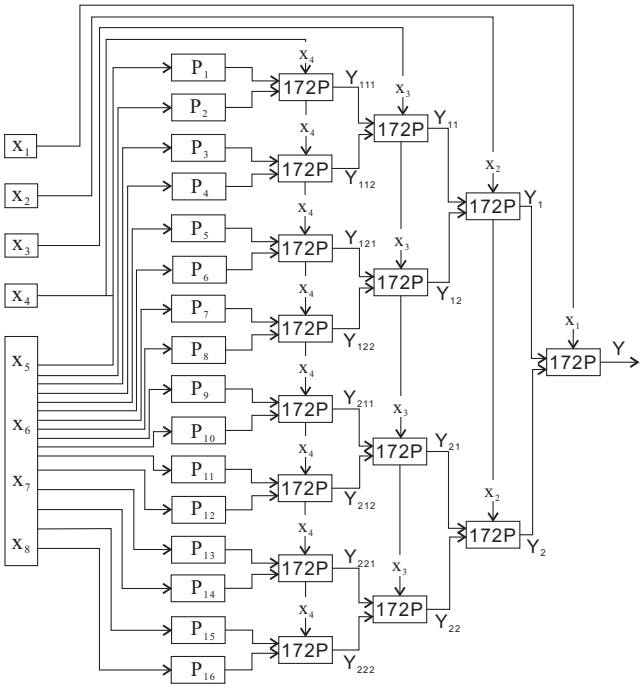

Figure 3. Networks implementing the Boolean function $y = f(x)$.

easily done using DNA-like learning algorithm.

For example, the weight-threshold values of 16 perceptrons in the network performing Boolean function $y = f_1(x)$ can be calculated, and they are shown in Table V.

IV. Conclusion

A new scheme for implementing S-box used in ciphers based on neural network is presented. Differing from the previous network models, the structure of the proposed network consists of multiple perceptrons (SLPs or MLPs) combining with a specific MLP ($172P$), and each perceptron only has a low number of input variables. More importantly, the network has the characteristics of massively parallel processing, and can be used to perform any S-box. In addition, it is convenient to train quickly the weight-threshold values of the Boolean function in the network through DNA-like learning algorithm. Further work will include testing some cryptographic properties of the S-box such as balance property, completeness criterion, strict avalanche criterion, non-linearity criterion and bit independence criterion etc..

Acknowledgment

This research was supported by the NSFC (Grants No. 11171084 and No. 60872093).

References

[1] M. N. A. Noughabi, and B. Sadeghiyan, Design of S-box Based on Neural Network, International Conference on Electronics and Information Engineering (ICEIE), Vol. 2, pp. 172-178, 2010.

[2] D. Pointcheval, Neural Networks and Their Cryptographic Applications, in Proc. of Eurocode, pp. 183-193, 1994.

[3] A. G. Bafghi, R. Safabakhsh, and B. Sadeghiyan, Finding the differential characteristics of block ciphers with neural networks, international journal of Information Sciences, pp. 3118-3132, 2008.

[4] A. Webster, and S. Tavares, On the design of S-boxes, Advances in cryptology-CRYPT0'85, LNCS 218, pp. 523-534, Springer, 1986.

[5] R. Forre, The strict avalanche criterion: spectral properties of Boolean functions and an extended definition, Advances in cryptology: Proc of CRYPTOA'88, Berlin: Springer-Verlag, 1989.

[6] C. Adams, S. Tavares, The structured design of cryptographically good S-boxes, JCryptol, 3(1), pp. 27-41. 1990.

[7] X. Yi, S. Cheng, X. You, A method for obtaining cryptographically strong 8×8 S-boxes, Global telecommunications conference, GLOBECOMA'97, pp. 3-8, 1997.

[8] J. A. Anderson, An Introduction to Neural Networks. London, U.K.: MIT Press, 1995.

[9] F. Y. Chen, G. R. Chen, Q. He, G. He, and X. Xu, Universal perceptron and DNA-like learning algorithm for binary neural networks: Non-LSBF implementations, IEEE Trans. Neural Netw., Vol. 20, pp. 1293-1301, 2009.

[10] F. Y. Chen, G. R. Chen, G. He, X. Xu and Q. He, Universal perceptron and DNA-like learning algorithm for binary neural networks: LSBF and PBF Implementations, IEEE Trans. Neural Netw., Vol. 20, pp. 1645-1658, 2009.

[11] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.