Solving Parity–N Problems with Feedforward Neural Networks

Bodgan M. Wilamowski [1], David Hunter [1], and Aleksander Malinowski [2]
[1] Boise Graduate Center, University of Idaho
[2] ECE Department, Bradley University

Abstract - Several neural network architectures for computing parity problems are described. Feedforward networks with one hidden layer require N neurons in the hidden layer. If fully connected feedforward networks are considered, the number of neurons in the hidden layer is reduced to N/2. In the case of fully connected networks with neurons connected in cascade, the minimum number of hidden neurons is between $\log_2(N+1)-1$ and $\log_2(N+1)$. This paper also describes hybrid neuron architectures with linear and threshold-like activation functions. These hybrid architectures require the least number of weights. The described architectures are suitable for hardware implementation since the majority of weights equal +1 and weight multiplication is not required. The simplest network structures are pipeline architectures where all neurons and their weights are identical. All presented architectures and equations were verified with MATLAB code for parity-N problems as large as N=100.

I. Introduction

There has been a significant research effort made toward optimum design of threshold logic networks for many decades [1-8]. The results of this effort can be applied to unipolar neural networks based on the McCulloch and Pitts model [9], but cannot be applied to networks with bipolar neurons. There is also a parallel effort by many researchers to solve parity-N problems using "neurons" with special activation functions [10-14]. However, the work presented here is limited to neural networks with classical McCulloch-Pitts neurons or neurons with sigmoid-like activation functions.

In 1961, Minnink [1] showed that threshold networks with one hidden layer require N hidden threshold units to solve the parity-N problem. Stork and Allen [14] reduced this problem to two hidden units with diode-like activation functions. Fung and Li [15] also demonstrated that the minimum size of the hidden layer required to solve the N-bit parity problem is N neurons. If the network is fully connected (output neuron is also directly connected to inputs) then, as shown by Minnink [1], the number of hidden threshold units can be reduced by half. In this case, only N/2 neurons are required in the hidden layer for the parity–N problem. More recently, Paturi and Saks [3] showed that only $N/\log_2 N$ neurons are required. Siu, Roychowdhury, and Kailath [6] showed that when one more hidden layer is introduced, the total number of hidden units could be only $2\sqrt{N}$. Cotofana and Vassiliadis [8] demonstrated many compact neural network architectures. They derived an asymptotic bound for O(log N) for the neural realization size and depth for a larger class of symmetric functions. They also stated that "...the realization of generic symmetric functions, possibly O(log N) depth and size network still remain open and subject of further research..." This paper addresses this issue and demonstrates neural networks for parity-N problems with the total number of neurons equal to $\log_2 N$. We also demonstrate several modifications of these minimal networks where the number of weights is minimal too. In contrast to all previously published results, we demonstrate both unipolar and bipolar implementations of parity-N networks.

II. Solving Parity-N Problems Using Boolean Logic

The XOR and parity-N problems are used very frequently in digital systems. For example, conversion of Gray code to binary code uses a chain of XOR operators. Parity-N circuits are essential for error detection and correction. Digital additions and multiplications also require parity-N circuits. Parity-N systems are often used in digital transmission to detect errors or to detect hardware failures in digital memory.

Logic function implementation of XOR and XNOR gates is more complicated than implementation of NAND and NOR gates. The most common XOR realizations require several digital primitives. Parity-N problems are usually implemented in digital system by cascading XOR gates. This solution for the parity-N problem requires N-1 XOR gates. Circuits for parity-N computations consist of several layers and introduce significant delays.

Franco and Cannas [17] introduced modular neural networks suitable to solve parity problems. Their concept basically follows the traditional digital approach where parity-N problems can be solved using cascade connected XOR blocks. Two advantages of their architectures are a limited number of inputs for each neuron and an error surface with less local minima. These architectures lead to multi-layer networks and a larger number of neurons than in other architectures.

III. XOR and Parity-3 Problems in Neural Network Implementations.

Both the XOR and parity-3 problems can be visually illustrates in two and three dimensions respectively as shown in Fig. 1. For simplicity purposes, let us consider hard threshold bipolar neurons with signal levels –1 and + 1. As shown in Fig. 1(a), the hidden neuron separation lines are described by:

0-7803-7898-9/03/$17.00 ©2003 IEEE

$$\frac{x}{1}+\frac{y}{1}>1 \quad \text{and} \quad \frac{x}{-1}+\frac{y}{-1}<1 \tag{1}$$

or

$$x+y-1>0 \quad \text{and} \quad x+y+1>0 \tag{2}$$

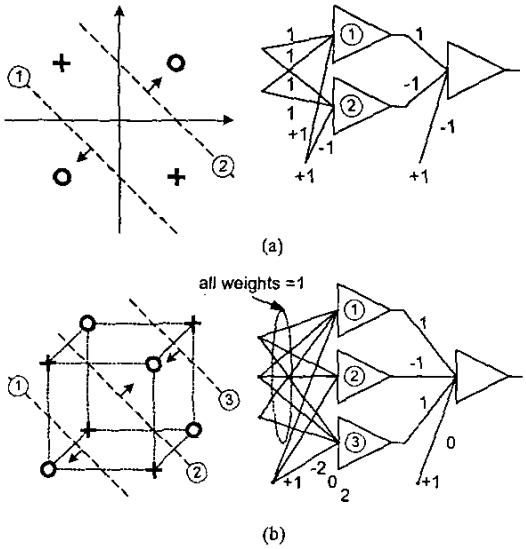


Fig. 1. Graphical interpretation of pattern separation by hidden layer and network implementation for (a) the XOR problem and (b) Parity-3 problem.

One may notice that for the XOR problem:

(1) All weights in the hidden layer connecied to inputs are equal to +1.
(2) Biasing weights for the two hidden neurons are +1 and −1. This fulfills (1).
(3) Weights for the output neuron are +1 and −1 and the bias is −1.

In the case of the Parity-3 problem (see Fig. 1(b)), the equations for the neurons in the hidden layer are given by:

$$x+y+z-2>0 \tag{3}$$

$$x+y+z+0>0 \tag{4}$$

$$x+y+z+2>0 \tag{5}$$

Please notice that (3) is only fulfilled when all inputs are +1, (4) is fulfilled when at least two inputs are +1, and (5) is fulfilled if one or more inputs are +1. In other words, the hidden neurons are counting the number of ones on the inputs. Neuron 1 only responds if there are three ones on the input. Neuron 2 responds if there are two or more ones on the inputs, and neuron 3 responds if there is a one on any input.
For the parity-3 problem, one may notice that:

(1) All weights in the hidden layer connected to inputs are equal to +1.
(2) Biasing weights for the three hidden neurons are +2, 0, and −2. This fulfills (3), (4), and (5).
(3) Weights for the output neuron are +1, -1, and +1 and the bias is 0.

The same architectures can be used for bipolar neurons, but the weights have to be transformed accordingly. For bipolar neurons:

$$net=\sum_{i=1}^{N} w_i x_i^{bip}+w_{N+1} \quad for \quad x\in(-1,+1) \tag{6}$$

The bipolar signal x_{bip} (in −1 to +1 range) can be computed as a function of the unipolar signal x_{uni} (in 0 to +1 range):

$$x^{bip}=2x^{uni}-1 \tag{7}$$

By inserting (7) into (6) one may obtain:

$$net=\sum_{i=1}^{N} 2w_i x_i^{uni}-\sum_{i=1}^{N} w_i+w_{N+1} \quad for \quad x\in(0,+1) \tag{8}$$

The division of both sides of (8) by 2 leads to:

$$0.5net=\sum_{i=1}^{N} w_i x_i^{uni}+0.5\left(-\sum_{i=1}^{N} w_i+w_{N+1}\right) \quad for \quad x\in(0,+1) \tag{9}$$

Therefore, to transform bipolar networks into unipolar networks, only the bias weights (thresholds) have to be recalculated. The input weights remain the same.

$$w_i^{uni}=w_i^{bip} \quad for \quad i=1,2,\cdots N \tag{10}$$

$$w_{N+1}^{uni}=0.5\left(w_{N+1}^{bip}-\sum_{i=1}^{N} w_i^{bip}\right) \tag{11}$$

Using (10) and (11), the bipolar circuits from Fig. 1 can be transformed to the unipolar circuits shown in Fig. 2.

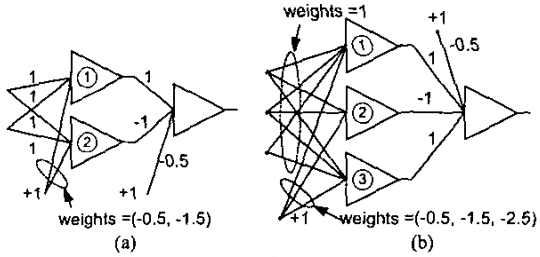
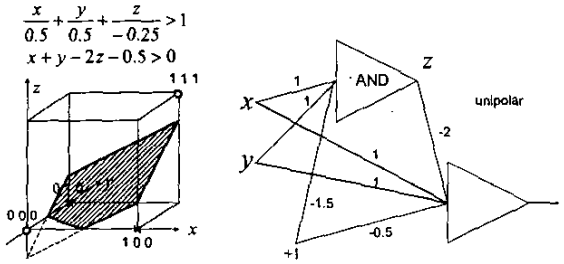


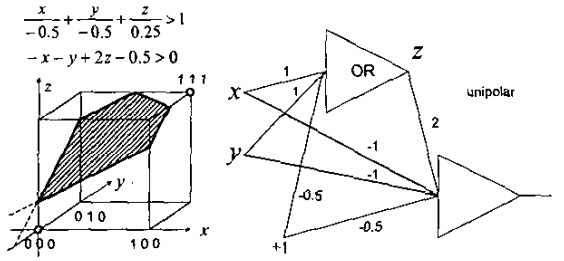
Fig. 2. Unipolar implementations of (a) XOR and (b) Parity-3 problems.

By using a fully connected network, the number of neurons in the hidden layer can be reduced. Fig. 3 and Fig. 4 illustrate two implementations of the XOR problem using only one neuron in the hidden layer. These circuits can be easily derived from functional link networks [16-17]. In the first circuit, the AND operator is used as the nonlinear element while in the second circuit, the OR operator is used as the nonlinear element.



(a) (b)

Fig 3. XOR implemetation with one neuron in the hidden layer performing AND operation (a) graphical interpretations and (b) diagram.



(a) (b)

Fig 4. XOR implementation with one neuron in the hidden layer performing OR operation (a) graphical interpretations and (b) diagram.

This time, the graphical interpretations of the circuits were done for unipolar neurons. These unipolar circuits can be transformed into bipolar ones using the following formulas:

$$w_i^{bip} = w_i^{uni} \quad for \quad i = 1,2,\cdots N \tag{12}$$

$$w_{N+1}^{bip} = 2w_{N+1}^{uni} + \sum_{i=1}^{N} w_i^{uni} \tag{13}$$

These formulas are derived by a method similar to (10) and (11).

IV. Solution for Parity-N Problem in Networks with One Hidden Layer

Solutions presented in Fig. 1 for the XOR and parity-3 problems can be generalized to parity-N cases. The values for the hidden neurons are given by:

$$net = (2i - N) + (N + 1 - 2j)$$

where:

$(2i - N)$ is the sum of inputs

$(N + 1 - 2j)$ is the bias (14)

N is total number of inputs

i is the number of inputs with +1 signal

j is the hidden neuron number

Please notice that:

$$net = \begin{cases} +1 & for \quad i \geq j \\ -1 & for \quad i < j \end{cases} \tag{15}$$

Therefore, the first neuron only responds if all N inputs are +1. The *j-th* neuron only responds if more than *j-1* inputs are activated with +1. For the *N-th* neuron, no inputs must be activated. As one can see from (14), the bias for the *j-th* neuron is:

$$bias_j = 2j - N - 1 \tag{16}$$

The output neuron performs the AND operation on all outputs of odd neurons with negation from all outputs of even neurons. This may be accomplished by assigning +1 weight values to all inputs coming from neurons with odd numbers and –1 weight values to the remaining inputs.

For the parity-N problem with layered neural networks containing one hidden layer, the weight calculations for the hidden neurons are:

$$w_{i,j} = 1 \quad for \quad i,j = 1,2,\cdots N \tag{17}$$

$$w_{N+1,j} = 2j - N - 1 \quad for \quad j = 1,2,\cdots N \tag{18}$$

While weights for the output neurons are:

$$v_1 = 2\,\mathrm{mod}_2(N) - 1 \tag{19}$$

$$v_j = -v_{j-1} \quad for \quad j = 2,3,\cdots N \tag{20}$$

$$v_{N+1} = \mathrm{mod}_2(N) - 1 \tag{21}$$

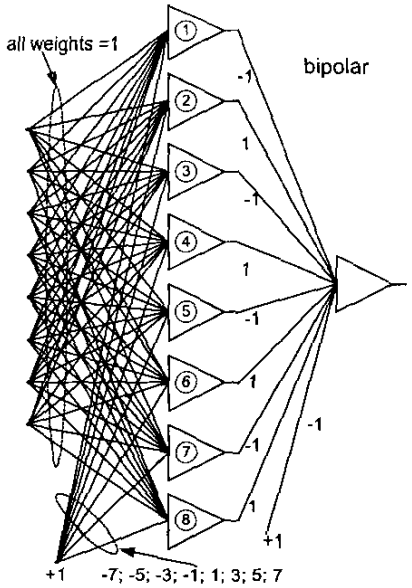


Fig. 5. Layered bipolar neural network with one hidden layer for the parity-8 problem.

For example, the architecture for the parity-8 problem with bipolar neurons is shown in Fig. 5. The same architecture can be used for a unipolar network. In this case:

$$w_{N+1,j} = 0.5 - j \quad for \quad j = 1,2,\cdots N$$
$$\overset{for\ N=8}{\Rightarrow} (-0.5,-1.5,-2.5,-3.5,-4.5,-5.5,-6.5,-7.5) \tag{22}$$

$$v_{N+1} = 0.5 - \mathrm{mod}_2(N) \overset{for\ N=8}{\Rightarrow} 0.5 \tag{23}$$

Please notice that in a layered feed forward network (no direct connections between inputs and the output neuron) for the parity-N problem, the number of hidden neurons is equal to N. The network complexity can be reduced if a fully connected network is considered. This issue is discussed in the next section.

V. Fully Connected Network with One Hidden Layer

The XOR architecture shown in Fig. 4 can be expanded for parity-N problems. Please notice that parity-N problems are symmetrical. This means that when inputs are mutually switched, the output remains the same. In other words, it is not important which inputs are excited, only that the total number of excited inputs is the same. Figure 6 shows the architecture for the parity-5 problem with two hidden unipolar neurons. Fig. 7 is a table that describes the state of the network for all possible input combinations.

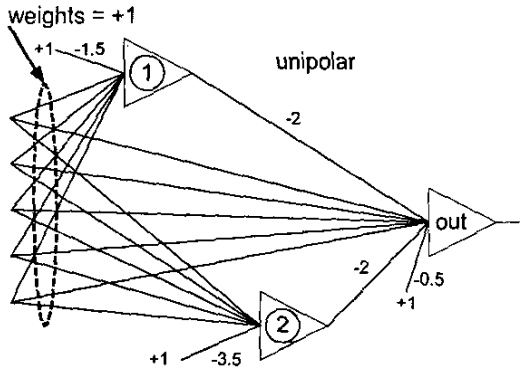


Fig. 6. Parity-5 implemented with 2 hidden neurons

j number of ones on inputs	o_1 neuron 1 output	$j - 2{\cdot}o_1$	o_2 neuron 2 output	$j - 2{\cdot}o1 - 2{*}o2$
0	0	0	0	0
1	0	1	0	1
2	1	0	0	0
3	1	1	0	1
4	1	2	1	0
5	1	3	1	1

Fig. 7. Table describing all possible states of network from Fig. 6

Similar architectures can be used for bipolar networks. The network diagram for the parity-11 problem is shown in Fig. 8 and the corresponding table describing all possible states is shown in Fig. 9.

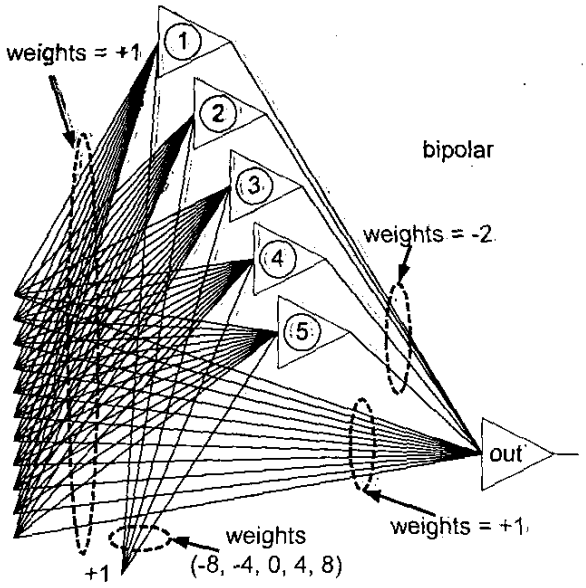


Fig. 8. Parity-11 implemented in fully connected bipolar neural networks with five neurons in the hidden layer.

n number of ones on inputs	net from inputs	o_1 neuron 1 output	o_2 neuron 2 output	o_3 neuron 3 output	o_4 neuron 4 output	o_5 neuron 5 output	$2{*}net$ from hidden outputs	net of output neuron
0	-11	-1	-1	-1	-1	-1	-10	-1
1	-9	-1	-1	-1	-1	-1	-10	+1
2	-7	1	-1	-1	-1	-1	-6	-1
3	-5	1	-1	-1	-1	-1	-6	+1
4	-3	1	1	1	-1	-1	-2	-1
5	-1	1	1	1	-1	-1	-2	+1
6	1	1	1	1	1	-1	2	-1
7	3	1	1	1	1	-1	2	+1
8	5	1	1	1	1	1	6	-1
9	7	1	1	1	1	1	6	+1
10	9	1	1	1	1	1	10	-1
11	11	1	1	1	1	1	10	+1

Fig. 9. Parity-11 case with the net and out values calculated for the bipolar architecture in Fig. 8.

Notice that the examples shown in Fig. 6 and Fig. 8 can be generalized for any value of N. Both bipolar and unipolar implementations are possible. Weights associated with all inputs are equal to +1. The bias weights for the hidden neurons, j, are:

$$w_{N+1,j} = 2(2j - J - 1) \quad for \quad j = 1,2,\cdots J \tag{24}$$

The weights for the output neuron are:

$$v_j = -2 \quad for \quad j = 1,2,\cdots J \tag{25}$$

VI. Fully Connected Network with One Neuron Per Layer

By connecting neurons in cascade, the number of required neurons can be significantly reduced. Fig. 10 shows the architecture for the parity-15 problem. Notice that only three hidden neurons are needed for this complex problem.

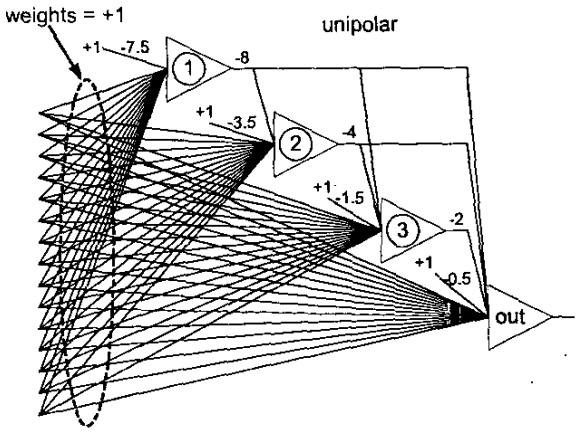


Fig. 10. Parity-15 implemented with 4 neurons in one cascade

n number of *ones* on inputs	*out* of neuron 1 Th=7.5	*net* of neuron 2	*out* of neuron 2 Th=3.5	*net* of neuron 3	*out* of neuron 3 Th=1.5	*net* of neuron 4	*out* of neuron 3 Th=0.5
0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	1
2	0	2	0	2	1	0	0
3	0	3	0	3	1	1	1
4	0	4	1	0	0	0	0
5	0	5	1	1	0	1	1
6	0	6	1	2	1	0	0
7	0	7	1	3	1	1	1
8	1	0	0	0	0	0	0
9	1	1	0	1	0	1	1
10	1	2	0	2	1	0	0
11	1	3	0	3	1	1	1
12	1	4	1	0	0	0	0
13	1	5	1	1	0	1	1
14	1	6	1	2	1	0	0
15	1	7	1	3	1	1	1

Fig. 11. State table for parity-15 with three hidden unipolar neurons in cascade connection as show in Fig. 10.

VII. Hybrid Networks

One may notice that the networks described in previous sections contain a large number of weights (see Fig. 8 and 10). These networks can be further simplified by introducing a linear neuron, which operates as a simple summator of the incoming signals. Figure 10 can be reduced to the diagram in Fig. 12.

Further network simplification is possible when each neuron is considered as a composition of a summator and a threshold unit. Figure 13 shows a pipline network architecture where all hidden neurons are identical and only have two weights. The connection coming from the summator of the previous neuron always has a weight of +2. The connection coming from the output of the previous neuron always has a weight of $-(N+1)/2$ (for parity 16 this is –8). The state-table for the bipolar pipeline architecture is shown in Fig. 14.

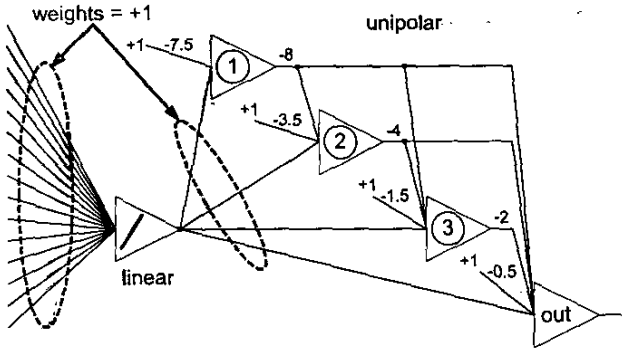


Fig. 12. Parity-15 implemented in hybride architecture with 5 neurons in one cascade and one neuron with a linear activation function.

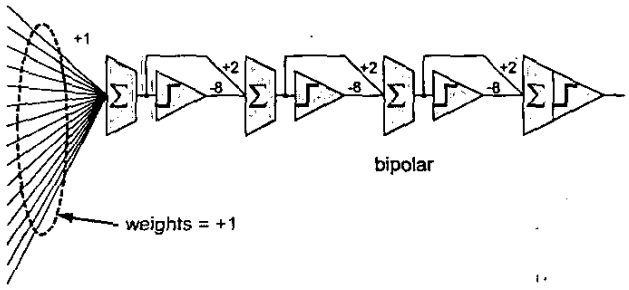


Fig. 13 Parity-15 implemented with 4 bipolar neurons in pipeline hybrid architecture with identical neurons and weights.

net of neuron 1	*out* of neuron 1 Th=0	*net* of neuron 2	*out* of neuron 2 Th=0	*net* of neuron 3	*out* of neuron 3 Th=0	*net* of neuron 4	*out* of neuron 3 Th=0
-15	-1	-14	-1	-3	-1	-1	-1
-13	-1	-10	-1	-1	-1	1	+1
-11	-1	-6	-1	1	+1	-1	-1
-9	-1	-2	-1	3	+1	1	+1
-7	-1	1	+1	-3	-1	-1	-1
-5	-1	3	+1	-1	-1	1	+1
-3	-1	5	+1	1	+1	-1	-1
-1	-1	7	+1	3	+1	1	+1
1	+1	-7	-1	-3	-1	-1	-1
3	+1	-5	-1	-1	-1	1	+1
5	+1	-3	-1	1	+1	-1	-1
7	+1	-1	-1	3	+1	1	+1
9	+1	1	+1	-3	-1	-1	-1
11	+1	3	+1	-1	-1	1	+1
13	+1	5	+1	1	+1	-1	-1
15	+1	7	+1	3	+1	1	+1

Fig. 14 State-table for parity-15 implemented with 4 neurons in pipeline architecture with identical neurons and weights.

VIII. EXPERIMENTS

It is not practical to test the described neural network architectures for all possible patterns. For example, if N=24, there are 2^{24} = 16,777,216 24-bit patterns to be applied to the system for full verification. The testing procedure can be significantly shortened. To verify correctness of the network, it is enough to apply N patterns (24 patterns for the given example).

Notice that the number of patterns required to test a network depends on the network architecture. To verify the architecture, there is no need to apply all possible patterns. It is enough to:

(A) Prove that the same response will be obtained for the same number of ones at the input, no matter which inputs are activated.

(B) Apply only N test patterns with a different number of ones in each pattern:

(1,0,0,....0), (1,1,0,....0), (1,1,1,....0), ...,(1,1,1,....1),

The proof (A) is very simple. Notice that the weights for all inputs are the same, +1. Therefore, inputs are exchangeable. This means that only the number of +1's and −1's is important, not their association with a particular input.

IX. CONCLUSION

Several neural network architectures for computing parity problems were described and tested. All architectures can be implemented with both bipolar and unipolar neurons. Feedforward networks with one hidden layer as described in Section IV require N neurons in the hidden layer. If a fully connected feed forward network is considered (Section V), then the number of neurons in the hidden layer is reduced to N/2. In the case of a fully connected network with neurons connected in cascade (Section VI), the minimum number of hidden neurons is between $\log_2(N+1)-1$ and $\log_2(N+1)$. The paper described hybrid neuron architectures with linear and threshold-like activation functions. These hybrid architectures (Section VII) require the least number of connections. The described architectures are all suitable for hardware implementation since the majority of the weights are equal to +1 and a weight multiplication process is not required. The simplest network structure is the pipeline architecture where all neurons and their weights are identical. All presented architectures and equations were verified with MATLAB code for parity-N problems as large as N=100.

Acknowledgment

This work was supported by NSF-Idaho EPSCoR and National Science Foundation grant EPS-0132626

REFERENCES

[1] R. C. Minnick, "Linear-input logic," *IRE Trans. Electronic* ., vol. EC-10, pp. 6-16, Mar. 1961.

[2] S. Muroga, "The principle of majority decision elements and complexity of their circuits," *Proc. Int. Conf Inform. Processing,* June 1959, pp. 400-407.

[3] R. Paturi and M. Saks, "On threshold circuits for parity," in *IEEE Symp. Foundation of Comput. Sci.,* pp. 397-404, Oct. 1990.

[4] S. Patarnello and P. Carnevali, "Learning networks of neurons with Boolean logic," *Europhys. Lett.,* vol. 4, pp. 503-508, 1987.

[5] C. V. den Broeck and R. Kawai, "Learning in feedforward Boolean networks," *Phys. Rev. A,* vol. 42, no 10, pp. 6210-6218, 1990.

[6] K. Y. Siu, V. Roychowdhury, and T. Kailath, "Depth-size tradeofd for neural computation," *IEEE Trans. Comput.,* vol 40, pp. 1402-1412, Dec. 1991.

[7] R. Impagliazzo, R. Paturi, and M. E. Saks, "Size-depth tradeoffs for threshold circuits," *SIAM J. Comput.,* vol 26, no. 3, pp. 693-707, 1997.

[8] S. Cotofana and S. Vassiliadis, "Periodic symmetric functions, serial additions, and multiplication with neural networks," *IEEE Trans. Neural Networks*, vol. 9, pp. 1118-1128, Nov. 1998.

[9] W. S. McCulloch and W. H. Pitts, "A logical calculus of the ideas imminent in nervous activity," *Bull Math. Biophy.* vol. **5**, pp 115-133, 1943.

[10] D. A. Brown, "N-Bit Parity Networks," *Neural Networks* vol. 6, pp. 607—608, 1993

[11] M. E. Hohil, D. Liu, and S. H. Smith, "Solving the N-bit parity problem using neural networks," *Neural Networks,* vol. 12, pp1321-1323, 1999.

[12] E. Lavretsky, "On the exact solution of the parity-N problem using ordered neural networks, *Neural Networks,* vol. 13 pp. 643-649, 2000.

[13] J. M. Minor, "Parity with two layer feedforward nets," *Neural Networks*, vol. 6, pp. 705-707, 1993

[14] D. G. Stork and J. D. Allen, "How to solve the N-bit parity problem with two hidden units" *Neural Networks*, vol. 5, pp. 923-926, 1992

[15] J. Hertz, a. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation.* Reading, MA: Addison-Wesley, 1991.

[16] N. J. Nilson, *Learning Machines: Foundations of Trainable Pattern Classifiers,* New York: McGraw Hill 1965.

[17] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Reading, Mass. Addison-Wesley Publishing Co. 1989