To be published as:
P. W. C. Prasad, A. Assi, and A. Beg, "Binary Decision Diagrams and Neural Networks," *The Journal of Supercomputing,* vol. 39, p. 20, March 2007.

Binary Decision Diagrams and Neural Networks

P.W.C. Prasad
College of Information Technology,
United Arab Emirates University,
UAE
prasadc@uaeu.ac.ae

Ali Assi
College of Information Technology,
United Arab Emirates University,
UAE
abeg@uaeu.ac.ae

Azam Beg
Department of Electrical Engineering,
United Arab Emirates University,
UAE
ali.assi@uaeu.ac.ae

Abstract

This paper describes a neural network approach that gives an estimation method for the space complexity of Binary Decision Diagrams (BDDs). A model has been developed to predict the complexity of digital circuits. The formal core of the developed neural network model (NNM) is a unique matrix for the complexity estimation over a set of BDDs derived from Boolean logic expressions with a given number of variables and Sum of Products (SOP) terms. Experimental results show good correlation between the theoretical results and those predicted by the NNM, which will give insights to the complexity of Very Large Scale Integration (VLSI)/Computer Aided Design (CAD) designs. The proposed model is capable of predicting the maximum BDD complexity (MaxBC) and the number of product terms (NPT) in the Boolean function that corresponds to the minimum BDD complexity (MinBC). This model provides an alternative way to predict the complexity of digital VLSI circuits.

Keywords: Binary Decision Diagrams, Neural Network, Complexity estimation, BDD complexity

1. Introduction

Logic level simulation is still one of the most often used operations in digital systems during both design and test stages [1]. With the rapid increase of the amount of circuitries on a single chip, there is a need for greater optimization and efficiency in the design process [2]. According to Moore's law [3] the number of transistors on a single chip doubles every year, and it has withstood the test of time since Gordon Moore made this observation in 1965. Boolean function representation has a direct influence on the computation time and space requirements of digital circuits and most of the problems in VLSI/CAD designs can be formulated in terms of Boolean functions. The efficiency of any method used depends on the complexity of Boolean functions [4]. Research on the complexity of Boolean functions in non-uniform computation models is now part of one of the most interesting and important areas in theoretical computer science [4], [5], [6]. Mathematicians and computer scientists kept trying to classify Boolean functions according to various complexity measures, such as the minimal size of circuits needed to compute specific functions [7].

During the last two decades, BDDs have gained great popularity as efficient method for representing Boolean functions [8], [9]. BDD in general is a direct acyclic graph representation of Boolean functions proposed by Akers [8] and further generalized by Bryant [9]. In many applications, the efficiency of BDD representations is determined by the size of the BDD defined as the number of nodes in the BDD for a given Boolean function f .

BDDs can be represented by algorithms that travel through all the nodes and edges of the directed graph in some order and therefore take polynomial time in the current size of the graph. However, when new BDDs are created, it might significantly increase the number of nodes in the BDD, depending on the node placement in the graph, which can lead to exponential memory and run time requirements [9] The choice of BDD variable order has a direct impact on the size of the BDD, and determining an optimal variable ordering is an NP-hard problem [10]. In general, it is hard to predict the effect of variable ordering on the BDD size, this requiring the trial of all possible ordering methods. It is also hard to find the best order for a given Boolean function. However, there are some observations that help in finding a good variable ordering.

The width of the BDD is defined as the maximum number of nodes at a certain level, where the level consist of nodes to which the same variable is assigned. The success of this technique has attracted researchers in the area of VLSI CAD systems [11], [12] and BDDs became very popular data structures. Evaluation of the space complexity of Boolean functions can be performed by determining the area of the BDD. Since the number of nodes in the BDD represents the space required for the function represented by the BDD, a function that produces higher number of nodes has higher space complexity than a function that produces lesser number of nodes. Building the complete BDD will increase the time complexity in the design process, more time is needed to implement, verify and test the design. Modeling is considered to be a time-efficient alternative to actual simulations and prototyping especially for non-linear and multi-variable systems. Predicting complexity of Boolean functions that represent a digital circuit could be a good indication to its feasibility prior to engaging in any further development and implementation. There have been a lot of research works [13], [14], [15] done on the estimation of combinational and sequential circuit parameters from the exact Boolean function describing the circuit. A mathematical model to predict the complexity of Boolean functions, XOR/XNOR min-terms and the path length of BDDs using empirical fit were introduced in papers [16], [17], [18], [19], [20], and here we propose an alternative way to tackle this problem using neural networks (NNs).

NNs have been showing amazing usefulness in the area of pattern recognition and prediction applications. Apart from this lot of research works has been done on the computational properties of neural networks [21], [22]. The measure of efficiency of the circuit have been addressed in relation with the area of circuit implementation [21],[23], where the complexity of Boolean functions is analyzed in terms of their implementation using different kind of circuits, from those with simple SOP, to feed forward neural network with threshold functions. In solving these problems the important contribution of the neural networks is their capacity for learning from experience. The main objective of this paper is to introduce a BDD complexity estimation methodology based on NNs. The resulting model will enable the design feasibility and performance to be analyzed without building the BDD. This model has produced competitive results against the mathematical prediction of the BDD complexity. In the second section, we provide background information pertaining to the BDD and NNs. Section three reviews the previous works done by the same authors on the estimation of BDD complexity. The proposed NNM for the estimation of BDD complexity is explained in the fourth, fifth and sixth sections. Section seven

To be published as:
P. W. C. Prasad, A. Assi, and A. Beg, "Binary Decision Diagrams and Neural Networks," *The Journal of Supercomputing,* vol. 39, p. 20, March 2007.
____
explains the NNM for the estimation of the MaxBC and the NPT of MinBC, follows with BDD Complexity comparison of NNM and Mathematical models for ISCAS Benchmark circuits in section eight. . Finally, we conclude our paper with our future developments in this research work.

2. Preliminaries

2.1 Binary Decision Diagram (BDD)

Basic definitions for binary decision diagrams are detailed in [5], [8], [9], [11]. The following is a summary of some of these definitions.

Definition 1: A BDD is a directed acyclic graph (DAG). The graph has two sink nodes labeled 0 and 1, representing the Boolean functions 0 and 1. Each non-sink node is labeled with a Boolean variable *v*, and has two out-edges labeled 1 (or *then*) and 0 (or *else*). Each non-sink node represents the Boolean function corresponding to its 1 edge if *v*=1, or the Boolean function corresponding to its 0 edge if v=0.

Definition 2: An *Ordered Binary Decision Diagram* (*OBDD*) is a BDD in which each variable is encountered no more than once in any path and always in the same order along each path.

Definition 3: A Reduced Ordered Binary Decision Diagram (ROBDD) is an OBDD where each node represents a distinct logic function. It has the following two properties:

(i) There are no redundant nodes in which both of the two edges leaving the node point to the same next node are present within the graph. If such a node exists, it is removed and the incoming edges redirected to the following node.

(ii) If two nodes point to two identical sub-graphs (i.e. isomorphic sub-graphs), then one sub–graph will be removed and the remaining one will be shared by the two nodes.

Variable Ordering

The size of a BDD is largely affected by the choice of the variable ordering [9], [11]. This is illustrated by the following example:

Example: Let $f = x_1 \cdot x_2 + ... + x_{2n-1} \cdot x_{2n}$. If the variable ordering is given by $(x_1, x_2, ..., x_n)$, i.e. $\pi(i) = x_i \forall_i$, the size of the resulting BDD is $2n$. The number of nodes in the graph varies linearly or exponentially depending on variable ordering. Fig. 1 shows the effect of variable ordering on the size of BDDs for the Boolean expression (1):

$$f = x_1 \cdot x_2 + x_1 \cdot \overline{x_2} \cdot x_3 \cdot x_4 + \overline{x_1} \cdot x_3 \cdot x_4 \qquad (1)$$

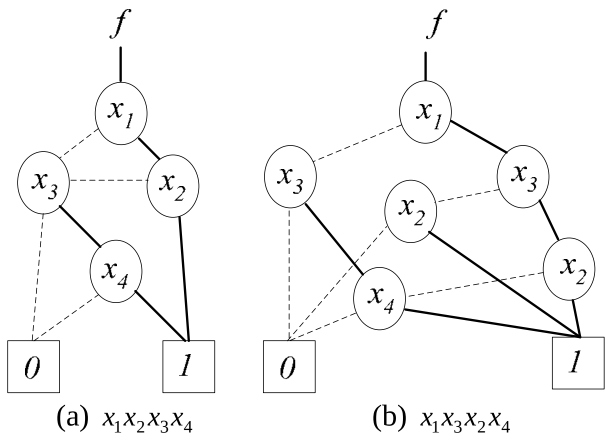

Figure 1 Effect of variable ordering on the size of BDDs, in term of number of nodes

2.2 Neural Networks (NNs)

NN mimic the ability of a human brain to find patterns and uncover hidden relationships in data. NNs can be more effective than statistical techniques for organizing data and predicting results, and are very efficient in modeling non-linear systems [24], [25], [26]. A NN is defined as a computational system comprising of simple but highly interconnected processing elements (PEs) (or neurons) (Figure 2) [27]. PEs are NN equivalents of biological neurons. Similarly, neural network interconnections are equivalents of synapses that connect a neuron to others. Information is processed by the PE's by dynamically responding to their inputs. Unlike conventional computers that process instruction and data stored in the memory in a sequential manner, the NNs produce outputs based on a weighted sum of all inputs in a parallel fashion.

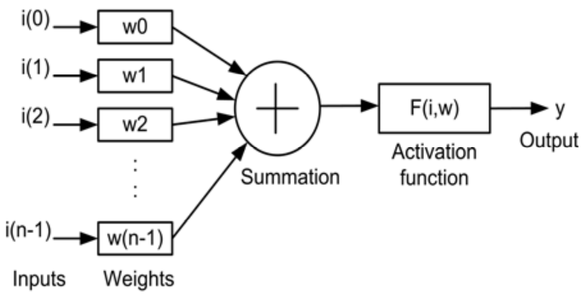

Figure 2. Processing element (PE) – building block of a neural network

In Figure 1 the inputs (*i(0)..i(n-1)*) to a PE are scaled with weights (*w(0)* .. *w(n-1)*) and summed up before being passed through an *activation function*. The activation function determines whether a PE *activates (fires)* or not. A s*igmoid* (non-linear) activation function has an s-shaped output between the limits [0, 1]. The function (2) is defined as [28]:

$$y = \frac{1}{(1+e^{-x})} \qquad (2)$$

Each input of an NN corresponds to a single attribute of the system being modeled. The output of the NN is the prediction we are trying to make.

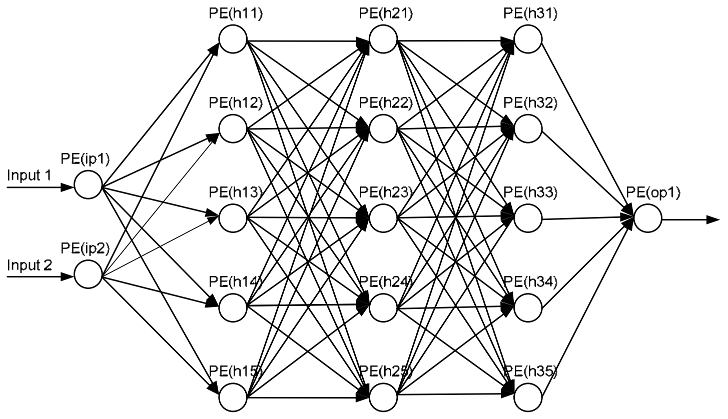

Figure 3. Topology of a simple 5-layer feed-forward neural network. The first is a 2-input layer followed by 3 hidden layers. A single output neuron makes the last layer.

Figure 3 shows the topology of a simple 5-layer *feed-forward* NN with 2 inputs and one output. The NN has 2 input neurons (*PE*(*ip1*), *PE*(*ip2*)), three hidden layers with 5 neurons each (*PE*(*hnm*) is the m^{th} neuron in n^{th} hidden layer), and one neuron in the output layer (*PE*(*op1*)) [24]. The NNM is fully-connected, meaning; all neurons in one layer connect to all neurons in the next layer.

NNs use different types of *learning* (or *training*) mechanisms, the most common of them being *supervised learning*. In this method of learning, a set of inputs is provided to the NN and its output is compared with the desired output. The difference between the actual and the desired outputs is used to adjust the weights (Figure 1) to different PEs in the network. The process of adjusting weights is repeated until the output falls within an acceptable range. To ensure a robust NN design, the set of input data and corresponding output data has to be chosen carefully. The input-output data set for an NN is called a *training set*. Additionally, special attention has to be paid to the formatting and scaling of the data for effective NN training [24]. The available data is divided into *training* and *validation sets*. An NN is only trained with the training set. Validation set is *run* on the NN to verify that the inputs are producing desirable outputs. If the validation phase produces large deviations, the training set or the network structure needs to be re-examined; re-training is required in this case [24].

3. Previous work

In this section we will briefly describe about background concept and results achieved in the area of the estimation of BDD complexity prior to introducing our NNM.

3.1 Relation between the Size of a Boolean function and the BDD Complexity

The complexity of the ROBDD mainly depends on the number of nodes represented by the BDD. An experiment was done in [16], [17] to analyze the complexity variation in BDDs i.e. the relation between the number of product terms and the number of nodes for any number of variables. The experimental and equation graph (Figure 4) shows that the complexity of the BDD can be modeled mathematically by (3).

$$NN = \alpha \cdot NPT^{\beta} \cdot e^{(-NPT \cdot \gamma)} + 1 \qquad (3)$$

where, *NN* is the number of nodes that represents the complexity of the BDD, *NPT* is the number of non-repeating product terms in the Boolean function, α , β and γ are three constants. Using curve fitting techniques, the variations of α, β and γ were mathematically modeled and represented by the following equations (4), (5) and (6).

$$\alpha = 0.9855 \cdot e^{(0.063 \cdot NV^{1.51})} \qquad (4)$$

$$\beta = 1.031149 \cdot e^{(-0.01551933 \cdot NV)} + 67.2072 \cdot e^{(-1.2985 \cdot NV)} \qquad (5)$$

$$\gamma = 0.962297281 \cdot e^{(-0.4187691 \cdot NV)} + 41.9723 \cdot e^{(-1.5072 \cdot NV)} \qquad (6)$$

Where, *NV* is the Number of Variables.

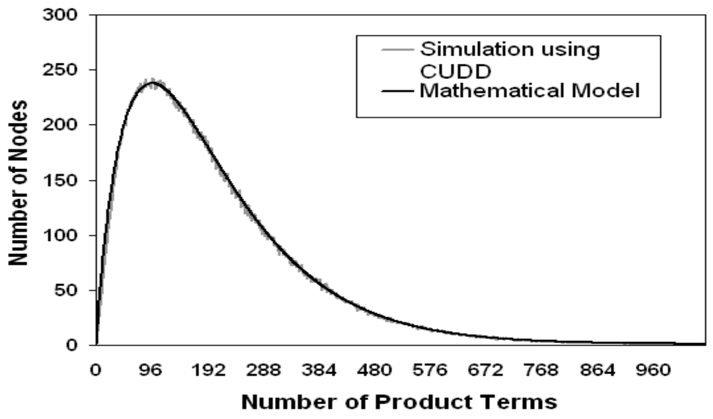

Figure 4. Simulation vs. mathematically predicted BDD complexity for 11 variables

4. Analysis of Boolean Function Complexity

For each variable count *n* between 1 and 14 inclusive and for each term count between 1 and 2^n-1, 100 SOP terms were randomly generated and the Colorado University Decision Diagram (CUDD) package [29] was used to determine the BDD complexity. This process was repeated until the BDD complexities (i.e. number of nodes) became 1. Then the experimental graphs for BDD complexity (Example: for 10 variables, as shown in Figure 5) were plotted against the product term count for each number of variables.

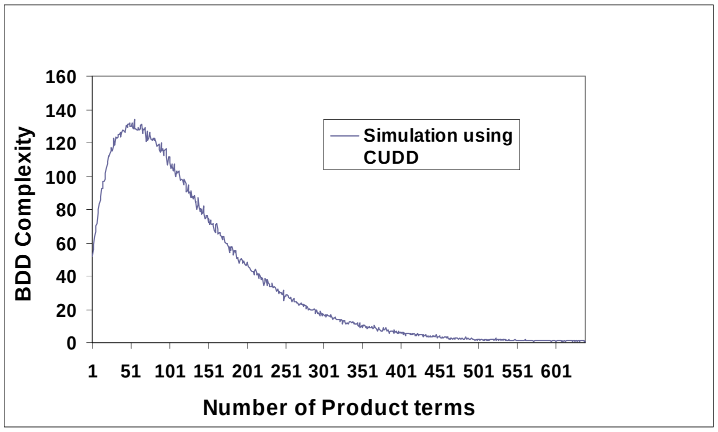

Figure 5. Simulation results for Boolean complexity for 10 variables

The above graphs indicate that the BDD complexity in general increases as the number of product terms increases. This is clear from the rising edge of the curve. At the end of the rising edge in the graph reaches a maximum complexity. This peak indicates the maximum BDD complexity (134) that any Boolean function with 10 variables can have independently of the number of product terms. Apart from that the peak also specifies the number of product terms (critical limit) of a Boolean function that leads to the MaxBC for any Boolean function with 10 variables.

The number of product terms that leads to the maximum for 10 variables is 54. If the number of product terms increases above the critical limit, as expected, the product terms starts to simplify and the BDD complexity will reduce. The BDD Complexity graph shown in Figure 5 indicates that as the number of product terms increases the complexity of the BDD decreases at a slower rate and ultimately reaches 1 node.

5. Neural Network Modeling Methodology

We have used here an NN-modeling software package called Brain Maker (Ver. 3.75 for MS Windows) [30] to create and test our NNMs. Brain Maker's back-propagation NNs were *fully connected*, meaning all inputs were connected to all hidden neurons, and all hidden neurons were connected to the outputs. The activation function for the hidden and output layer was a sigmoid function. The difference between the network's actual output and the desired output was treated as the error to be minimized. We stopped the NN training sessions, when the earliest of these conditions were met [30].

a) 98% of the facts were learnt with less than 5% mean squared error or
b) When training iterations count (epochs) reached 10000

For the sake of training ease, we chose to develop two separate models, one for predicting the BDD complexity and the other for predicting the MaxBC and the NPT for MinBC for a given number of variables.

6. Application of Neural Network to BDD Complexity Modeling

This section covers the definition and implementation of the Neural Network Model (NNM) for modeling the BDD complexity (Figure 6).

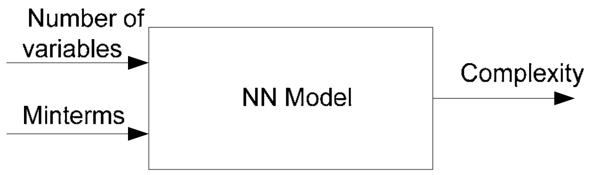

Figure 6. NNM for BDD complexity. The NNM has two inputs (*number of variables*, and *number of min-terms*) and one output (*BDD complexity*).

6.1 Data Collection and Processing

For the NNMs in this paper, the training and validation data sets were obtained by the experiment done in section 3.1. Pre-processing the training and validations sets takes a considerable amount of resources for a practical and reliably functioning NNM [26], [30]. In our research, the first data pre-processing step was to transform the data set in such a way that inputs have *equitable distribution of importance*. In other words, the larger absolute values of an input should not have more influence than the inputs with smaller magnitudes [31]. The need of such equitable distribution can be explained with the set of figures shown below. Figure 7 shows the *raw* (original) data for 2 to 14 variables. Notice that the plots for 2- to 9-variables are hardly visible when all variables are plotted on the same scale. If the data were presented to the NN for training in this case, only 10- to 14- variable cases could be learnt by the NN and 2- to 9-variables values could be ignored. So in order to provide similar importance to all variable values (2 to 14), we did a logarithmic transformation of the product terms (min-terms) and complexity (number of nodes) inputs. The resulting data is plotted in Figure 8. As we can see now all different plots (from 2 to 14 variables) are in similar ranges and make it easier for NN to learn them.

6.2 Training and Testing of NNM for BDD Complexity

In order to 'use' or 'run' a trained NN, de-normalization and de-transformation has to be done to restore the predicted outputs to the original ranges. Steps employed in 'training' and 'running' the networks are summarized here:

6.2.1 Steps for Training the NNM:
a) Take logarithm of actual values of the inputs and output
b) Train the NN with values from step (a)

6.2.2 Steps for Using/Running the NNM:
a) Take logarithm of the actual values of the input
b) Present the values from step (a) to the NNM
c) Apply anti-logarithm to the output of the NNM to get the actual result

To be published as:
P. W. C. Prasad, A. Assi, and A. Beg, "Binary Decision Diagrams and Neural Networks," *The Journal of Supercomputing,* vol. 39, p. 20, March 2007.

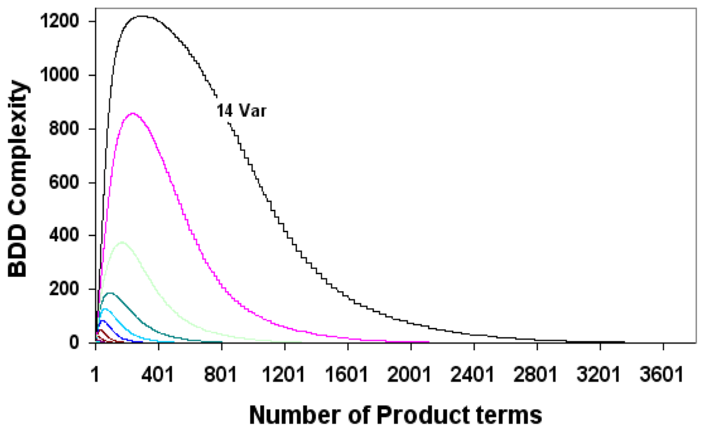

Figure 7. Raw (untransformed) data. Notice that the smaller variable (2, 3, etc.) curves are not as *visible* as the larger values.

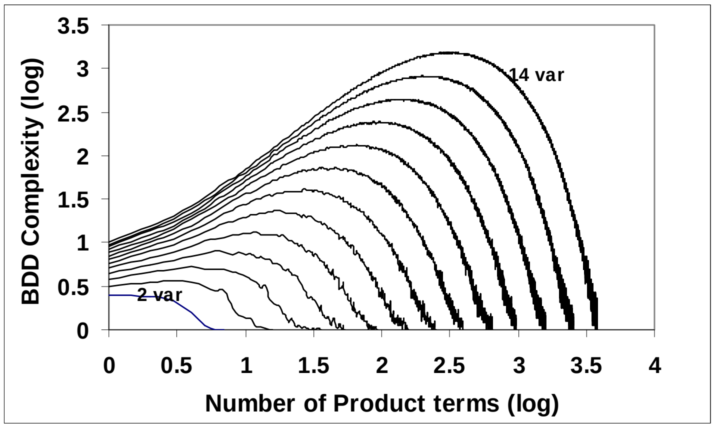

Figure 8. Log-scaled (transformed) data. All curves are now on a similar scaled which improves the NNM prediction accuracy.

We acquired a total of 19044 data sets (also called *facts/training facts*) during our simulations of Boolean Functions. 90% of the data sets (facts) were used as the training set, while the remaining 10% were used as the validation set. We stopped the NN training sessions, when 98% of the facts were learnt with less than 5% mean squared error [30].

A general rule is that as the number of hidden layers increases, the prediction performance goes up, but only up to a certain point, after which the NNM performance starts to deteriorate [30]. To find the optimum topologies for our NNMs, we experimented with up to 3 hidden layers; each layer consisted on a different number of neurons. The details of some of our NNMs experiments are listed in Table 1. The performance metric for an NNM was the "percentage of facts learnt with 95% (or more) accuracy". We chose a 5-layer NNM (#8 in the table) with 5 neurons in each of its hidden layers.

Table 1
Configuration & Training Statistics for BDD-Complexity NNMs *

CONFIGURATION						TRAINING STATISTICS			
No.	Input Layer Neurons	Hidden Layer 1 Neurons	Hidden Layer 2 Neurons	Hidden Layer 3 Neurons	Output Layer Neurons	Facts Learnt	Facts Not Learnt	% Facts Learnt	Epochs
1	2	10			1	12524	4789	72.3%	1047
2	2	20			1	16208	1105	93.6%	623
3	2	25			1	16059	1254	92.8%	745
4	2	30			1	15844	1469	91.5%	630
5	2	5	5		1	16889	424	97.6%	681
6	2	7	7		1	15261	2052	88.1%	2133
7	2	20	20		1	16987	326	98.1%	100
8	**2**	**5**	**5**	**5**	**1**	**17028**	**285**	**98.4%**	**98**
9	2	5	10	5	1	17049	264	98.5%	24
10	2	20	20	20	1	17079	234	98.6%	17

* Brain Maker training parameters: Training tolerance = 0.05; testing tolerance = 0.05; learning rate adjustment type = heuristic. (See [35] for detailed explanation of these settings).

This configuration provided nearly the same training accuracy as its much larger 3-layer counterparts (#9 and #10). The matrices containing weights for different layers of the chosen 5-5-5 neuron NNM (#5) are given in tables 2.1-2.4.

Table 2.1
Weight Matrix – Input Neuron Layer to Hidden Neuron Layer-1

	ip1	ip2
h11	0.915	3.255
h12	-6.639	-7.992
h13	3.511	7.981
h14	-5.674	7.983
h15	-6.618	-7.992

Table 2.2
Weight Matrix – Hidden Neuron Layer-1 to Hidden Layer-2

	h11	h12	h13	h14	h15
h21	2.811	-7.999	-1.437	-7.636	-7.999
h22	6.195	-7.996	-0.372	-7.999	-7.998
h23	-7.999	7.994	-7.949	1.305	7.999
h24	-4.474	-7.998	-7.339	-2.447	-7.681
h25	1.567	H	3.280	0.197	-7.999

Table 2.3
Weight Matrix – Hidden Neuron Layer-2 to Hidden Layer-3

	h21	h22	h23	h24	h25	h26
h31	-1.397	-2.336	-3.671	7.999	-2.518	5.084
h32	7.108	0.165	-7.999	1.441	4.130	-7.657
h33	-7.999	-7.999	5.342	-6.485	-1.831	-7.996
h34	-5.675	0.421	2.268	0.539	-2.678	5.324
h35	-0.130	3.669	-2.209	1.186	-0.938	0.138

Table 2.4

Weight Matrix – Hidden Neuron Layer-3 to Output Layer

	h31	**h32**	**h33**	**h34**	**h35**
op1	-4.417	6.338	7.999	-6.016	4.531

The weight matrices for the trained NNM (#8) Table 1 are shown in Tables 2.1-2.4. Refer to Figure 9 for details on different neuron layers. For example, weight in *ip1-h11* cell in the Table 2.1 refers to weight between the input "*ip1*" and "*h11*" neuron of the first hidden layer. Similarly, in the Table 2.2 the weight at the *h11-h21* location refers to the weight between the first layer neuron "*h11*" and the 2nd hidden layer "*h21*".

6.3 NN Modeling Results and Analysis

Due to the inherent nature of NNMs, the input values used for running an NNM should be kept somewhat close to, but not necessarily the same as, the input values in the training set. Any significant deviations of the running set from the training set can provide misleading results. We used an arbitrary set of values for number-of-variables and NPT and used the NNM to predict the number of nodes (BDD complexity).
Figure 9 indicates the comparison for experimental results and NNM predictions of BDD complexity for 10 variables. It can be inferred that the NNM result provides a very good approximation of the BDD complexity.

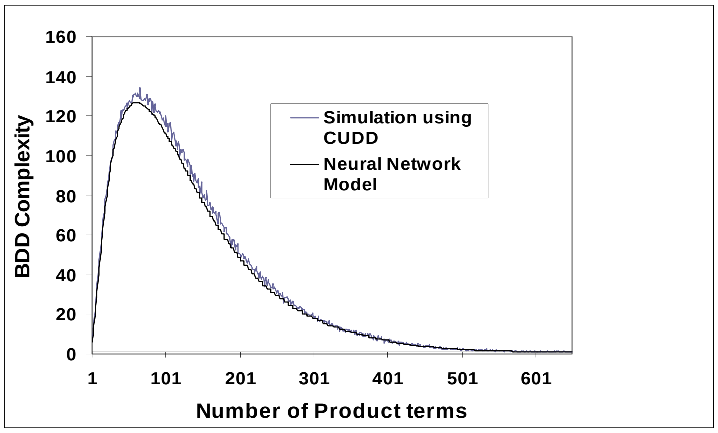

Figure 9. Complexity analyses of simulation and NNMs for 10 variables

The same work has been repeated for Boolean functions with 2 to 15 variables. Figure 10, 11, and 12 illustrate experimental and predicted NNM results for variables 7, 12 and 14 respectively.

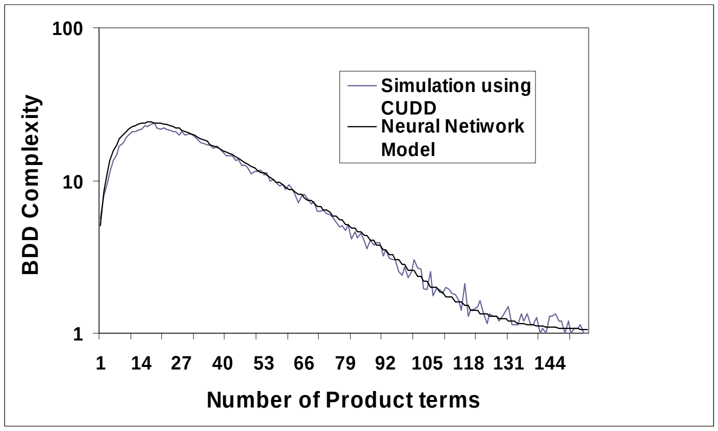

Figure 10. Complexity analyses of simulation and NNMs for 7 variables

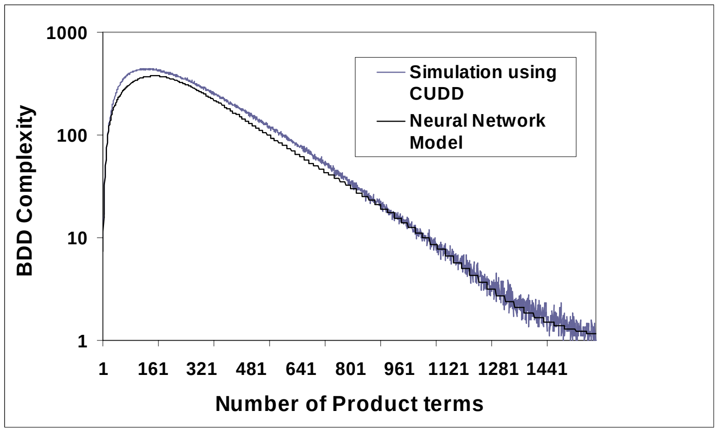

Figure 11. Complexity analyses of simulation and NNMs for 12 variables

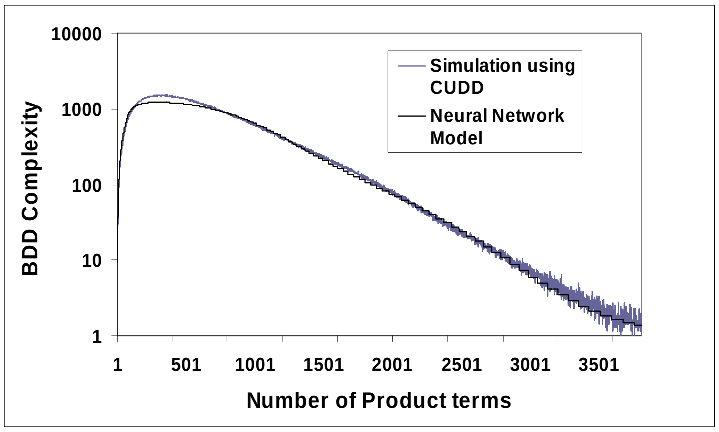

Figure 12. Complexity analyses of simulation and NNMs for 14 variables

Figure 13 shows the efficiency of the proposed NNM, which produces very close fit as the mathematical model [17] for the prediction of BDD complexity. It can be inferred that the NNM was able to match the experimental curve with minimum error for most of the Product terms.

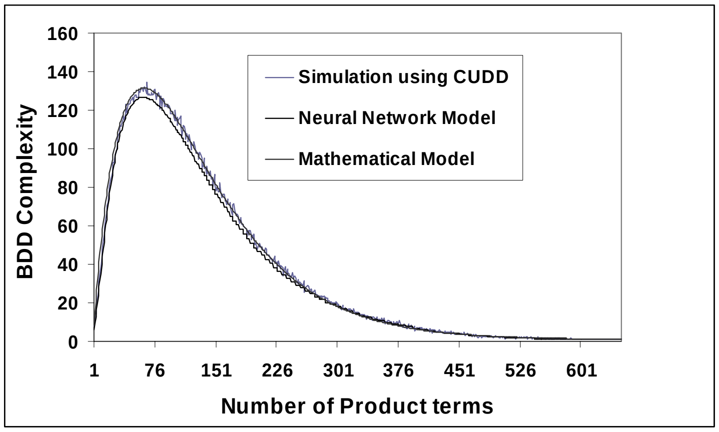

Figure 13. Comparison of actual simulations with mathematical and NN models

7. Application of Neural Networks to the Modeling of MaxBC and NPT of MinBC

This section covers the definition and implementation of the NNM for modeling the MaxBC and NPT of MinBC (Figure 14).

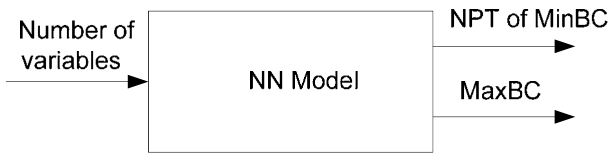

Figure 14. NNM for MaxBC and the NPT of MinBC. There is a single input (*number of variables*) to the NNM which predicts two values (*number of product terms for the lowest BDD complexity*, and the *maximum BDD complexity*).

Just like the NNM in the previous section, the training and validation data sets were obtained by the experiment done in section 5.

7.1 Training and Testing of NNM for BDD Complexity

For NN training purposes, we pre-processed the input data by taking its logarithm (the reason for doing so has been explained in section 6).
In summary the steps used to *train* and *run* the network are given here:

7.1.1 Steps for Training the NNM:
a) Take logarithm of actual values of the inputs and output
b) Train the NN with values from step (a)

7.1.2 Steps for Using/Running the NNM:
a) Take logarithm of the actual values of the input
b) Present the values from step (a) to the NNM
c) Apply anti-logarithm to the output of the NNM to get the actual result

To create the NNM, we only had a limited number of data sets, just 13 to be specific. Here again, we experimented with up to 3 hidden layers; each layer consisted of a different number of neurons. The details of some of our NNMs experiments are listed in Table 2.5.

Table 2.5
Configuration & Training statistics for MaxBC and MinBC

No.	Input Layer Neurons	Hidden Layer 1 Neurons	Hidden Layer 2 Neurons	Hidden Layer 3 Neurons	Output Layer Neurons	Facts Learnt	Facts Not Learnt	% Facts Learnt	Epochs
1	1	5			2	9	4	69.2%	10000
2	1	10			2	11	2	84.6%	10000
3	**1**	**15**			**2**	**13**	**0**	**100.0%**	**10000**
4	1	20			2	13	0	100.0%	10000
5	1	3	3		2	9	4	69.2%	10000
6	1	5	5		2	9	4	69.2%	10000
7	1	7	7		2	11	2	84.6%	10000
8	1	10	10		2	9	4	69.2%	10000
9	1	3	3	3	2	10	3	76.9%	10000
10	1	5	5	5	2	10	3	76.9%	10000
11	1	5	10	5	2	10	3	76.9%	10000

* Brain Maker training parameters: Training tolerance = 0.07; testing tolerance = 0.07; learning rate adjustment type = heuristic

We chose a 3-layer NNM (#3 in the table) with 15 neurons in its only hidden layer. This configuration was able to model all data sets with the desired accuracy. (#4 also trained with the same accuracy, but with larger number of neurons). The matrices containing weights for different layers of the chosen 1-5-2 neuron NNM (#3) are given in Table 2.6-2.7. For example, weight in *ip1*-*h1* cell in the Table 2.6 refers to weight between the inputs "*ip1*" and "*h1*" neuron of the hidden layers. Similarly in Table 2.7 the weight at the *h1*-*op2* location refers to the weight between the hidden "*h1*" neuron and the output neuron "*op2*".

Table 2.6
Weight Matrix – Input Neuron Layer to Hidden Neuron Layer

	ip1
h1	0.832
h2	-3.196
h3	-4.047
h4	-7.061
h5	-5.283
h6	-2.455
h7	0.228
h8	-3.753
h9	0.036
h10	-1.035
h11	3.762
h12	-3.082
h13	-0.986
h14	-1.950
h15	-2.382

Table 2.7
Weight Matrix – Hidden Neuron Layer to Output Neuron Layer

HIDDEN NEURON LAYER TO OUTPUT NEURON

	h1	**h2**	**h3**	**h4**	**h5**	**h6**	**h7**	**h8**
op1	0.602	-1.547	-3.286	-1.260	-0.899	-0.545	-1.768	-1.778
op2	1.112	-3.171	0.419	-0.566	-0.264	-2.824	2.053	-2.899
	h9	**h10**	**h11**	**h12**	**h13**	**h14**	**h15**	
op1	0.912	-0.056	1.113	-1.889	4.172	-1.300	0.242	
op2	-0.621	-0.297	0.151	-1.123	0.234	1.371	-1.167	

8. BDD Complexity Analysis use of Benchmark Circuits

The validated results for NNM for selected ISCAS benchmark circuits are tabulated in Table 1. The experimental results were obtained on a Pentium IV machine with 512 MB RAM running on Linux environment. Training of NNM with the experimental data has an up-front once only cost, and then the NNM is quickly run (within a few milliseconds or less) to predict the complexity of various functions with different number of variables and product terms. Running the models is generally faster than simulations, especially when larger benchmarks are involved (Hossain, et al., 2002).

The 1st Column indicates the ISCAS benchmark circuit name and the 2nd and 3rd columns are for the maximum number of input variables and number of output circuits for the respective benchmark circuit. In column 4, the actual BDD complexity for the benchmark circuits have been calculated using CUDD package. The NNM prediction was calculated for the each number of variables and number of product terms for the each respective benchmark circuits. The columns 5, 6 and 7 illustrate the BDD complexity prediction use of NNM, NNM prediction error with compare to the actual BDD complexity results and the relative error respectively. The mathematical model results for BDD complexity is given in column 8, follows with column 9 and 10 for Mathematical complexity error and relative error compared to the actual BDD complexity value, respectively.

Table 1
NNM results for ISCAS benchmark circuits

Circuit Name	Max Number of Input Variables	Number of Circuits	BDD Complexity (Nodes)						
			Actual Values	Neural Network Model			Mathematical Model		
				Predicted Value	Prediction Error	Relative Error	Predicted Value	Prediction Error	Relative Error
Apex4	9	19	1286	1137	-149	-0.116	1009	-277	-0.215
5xp1	7	10	90	100	10	0.111	85	-5	-0.056
apex7	48	55	344	349	5	0.015	289	-55	-0.160
Alu4	14	8	1162	1209	47	0.040	1426	264	0.227
B1	3	4	12	9	-3	-0.250	9	-3	-0.250
Cc	21	18	105	101	-4	-0.038	84	-21	-0.200
C8	28	17	149	157	8	0.054	189	40	0.268
decod	5	16	96	94	-2	-0.021	72	-24	-0.250
Clip	9	5	394	374	-20	-0.051	262	-132	-0.335
cm42a	3	13	50	46	-4	-0.080	31	-19	-0.380
cm138a	6	8	56	59	3	0.054	68	12	0.214
cm162a	11	6	66	79	13	0.197	122	56	0.848

cm163a	9	5	59	71	12	0.203	63	4	0.068
cm82a	5	3	19	16	-3	-0.158	15	-4	-0.211
Con1	6	2	18	17	-1	-0.056	14	-4	-0.222
Cu	14	11	89	75	-14	-0.157	74	-15	-0.169
majority	5	1	8	7	-1	-0.125	6	-2	-0.250
misex1	8	7	69	70	1	0.014	60	-9	-0.130
Rd53	5	3	24	18	-6	-0.250	19	-5	-0.208
Rd73	7	3	38	40	2	0.053	33	-5	-0.132
Rd84	8	4	54	41	-13	-0.241	24	-30	-0.556
Sao2	10	4	171	190	19	0.111	215	44	0.257
Sqrt8	8	4	43	45	2	0.047	46	3	0.070
Sct	14	15	177	215	38	0.215	213	36	0.203
squar5	5	8	57	49	-8	-0.140	52	-5	-0.088
X2	10	7	68	70	2	0.029	80	12	0.176
x1	25	35	557	732	175	0.314	691	134	0.241
xor5	5	1	6	5	-1	-0.167	5	-1	-0.167
I6	5	67	413	355	-58	-0.140	345	-68	-0.165
I7	6	67	493	598	105	0.213	501	8	0.016
Pm1	9	13	80	105	25	0.313	88	8	0.100
C17	4	2	10	8	-2	-0.200	9	-1	-0.100
Alu2	10	6	249	290	41	0.165	265	16	0.064
Cht	47	36	192	176	-16	-0.083	166	-26	-0.135
i3	132	6	138	186	48	0.348	173	35	0.254
i8	133	81	2498	3148	650	0.260	3482	984	0.394
cm150a	21	1	78	93	15	0.192	98	20	0.256
b12	15	9	97	106	9	0.093	131	34	0.351
cmb	16	4	59	61	2	0.034	65	6	0.102
b9	41	21	310	301	-9	-0.029	287	-23	-0.074
Total		605	RMS Error			**0.163**			**0.260**

The RMS relative error was computed as an estimation of the deviation of the measured from predicted values.

$$relative\ error = {(predicted\ value - CUDD\ measured\ value)}\Big/{CUDD\ measured\ value}$$

$$rms\ error = \sqrt{\frac{\sum_{i=1}^{n}(relative\ error_i)^2}{n}}$$

The computed deviation for Boolean function complexity estimation use of NNM for complete set of 610 circuits was 0.163, which indicates that the NNM provides a close match with the actual complexity values for the benchmark with maximum of 133 input variables. It can be inferred from the benchmark circuit validation, that all the benchmarks are mostly consist of product terms of 1-13 variables, even though the number of inputs goes up to much high than 14. It was also observed that most of the benchmarks do not have product terms which can produce the maximum Boolean function complexity for the variable of that product terms. The NNMs were more effective compared to the mathematical model RMS error of 0.260 for the same benchmark circuits.

It can be concluded from these results that the training set had a statistically significant number of samples to present to a neural network and it allowed NNM to be adequately trained within a user-define accuracy, therefore NNM was able to predict the BDD complexity for variables beyond the experimental data limits.

9. Conclusion

In this research work, we have proposed a new BDD complexity prediction methodology based on neural network as another alternative to the CUDD simulation and the mathematical models presented by the same authors. An advantage of this model is that it is a single integrated model for different number of variables and number of product terms. The results show the capabilities of training algorithms in neural networks, which produce a close match for the CUDD simulation with minimum errors of 4.22%, 1.32% and 0.39% for the calculation of the BDD complexity, the MaxBC and the NPT of MinBC respectively. Once NNMs had been developed, they could be used to conduct further experiments with different types of inputs, in a fraction of the time what a circuit simulator would take. The NNM was capable of providing useful clues about the complexity of the final design, which will leads to a great reduction in time complexity for digital circuit's designs. The NNM RMS error of 0.161 for ISCAS benchmark circuit justifies the efficiency of the NNM compared to the mathematical model. In light of the results, we conclude that the NNM proposed in this work could be a valuable tool for exploring the complex computational capabilities of neural network. We are currently exploring the extension of this work to other complexity applications.

References

[1] M. Alexander. Digital Logic Testing and Simulation, Chapter 2: "Combinational Logic Test," Harper and Row, New York, 1986.

[2] M. Thornton and V.S.S. Nair. Iterative Combinational Logic synthesis Techniques using Spectral Data. Technical report, Southern Methodist University, 1992.

[3] G. E. Moore. Progress in Digital Integrated Electronics. IEEE IEDM, pp. 11-13, 1975.

[4] I. Wegener. The Complexity of Boolean Functions. John Wiley and Sons Ltd,.

[5] C. Meinel and A. Slobodova. On the Complexity of constructing Optimal Ordered Binary Decision diagrams. Proc. of 19th Inter. Symposium on Mathematical Foundation of Computer Science, 515-524, 1994.

[6] S. Tani, K. Hamaguchi and S. Yajima. The Complexity of the Optimal Variable Ordering Problems of a Shared Binary Decision Diagram. IEICE Transactions on Information and Systems, Vol. 4, 271-281, 1996.

[7] M. Nemani, and F.N. Najm. High-level power estimation and the area complexity of Boolean functions. Proc. of IEEE Intl. Symposium on Low Power Electronics and Design, pp: 329-334, 1996.

[8] S. B. Akers. Binary Decision Diagram. IEEE Trans. Computers, Vol. 27, pp. 509-516, 1978.

[9] R. E. Bryant. Graph–Based Algorithm for Boolean Function Manipulation. IEEE Trans. Computers, Vol. 35, pp. 677-691, 1986.

[10] J.E. Harlow, and F. Brglez. Design of experiments and evaluation on of BDD ordering heuristics. Intl. J. Software Tools for Tech. Transfer., 3: 193-206, 2001

[11] S. Minato. Binary Decision diagrams and Applications for VLSICAD. Kluwer Academic Publishers, Dordrecht, 1995.

[12] K. Priyank. VLSI Logic Test, Validation and Verification, Properties & Applications of Binary Decision Diagrams. Lecture Notes, Department of Electrical and Computer Engineering University of Utah, Salt Lake City, UT 84112.

[13] P. E. Dunne, and W. van der Hoeke. Representation and Complexity in Boolean Games. proc. 9th European Conference on Logics in Artificial Intelligence, LNCS 3229, Springer-Verlag, pages 347-35, 2004.

[14] N. Ramalingam, S. Bhanja. Causal Probabilistic Input Dependency Learning for Switching model in VLSI Circuits. proceedings of ACM Great Lakes Symposium on VLSI, pp. 112-115, 2005.

[15] S. Bhanja, K. Lingasubramanian and N. Ranganathan. Estimation of Switching Activity in Sequential Circuits using Dynamic Bayesian Networks. proceedings of VLSI Design 2005, pp. 586-591, 2005.

[16] M. Raseen, P.W.C. Prasad and A. Assi. Mathematical Model to Predict the Number of Nodes in an ROBDD. The 47th IEEE Inter. Midwest Symposium on Circuit and Systems (MWSCAS), Vol. III, 431-434, 2004.

[17] M. Raseen, P.W.C. Prasad, and A. Assi. An Efficient Estimation of the ROBDD's Complexity. accepted for Publication in Integration - the VLSI journal, Elsevier Publication, May 2005

[18] P.W. C. Prasad , M. Raseen and S. M. N. A. Senanayake. XOR/XNOR Functional Behavior on ROBDD Representation. Proc. of The 14th IASTED Inter. Conf. on Applied Simulation and Modeling, pp. 115-119, 2005.

[19] P.W. C. Prasad , M. Raseen, A. Assi, B.I. Mills, and S. M. N. A. Senanayake. Evaluation time estimation for Pass Transistor Logic Circuits. accepted for publication in the proceedings in 3rd IEEE Inter. Workshop on Electronic Design, Test and Applications (DELTA'06), January 2006.

[20] A. Assi, P.W. C. Prasad, B. Mills, and A. El-chouemi. Empirical Analysis and Mathematical Representation of the Path Length Complexity in Binary Decision Diagrams. Journal of computer Science, Science Publications, Vol. 2(3), pp. 236-244, 2005.

[21] I. Parberry. Circuit Complexity and Neural Networks. MIT Press (1994).

[22] K. Y. Siu, V. P. Roychowdhury and T. Kailath, Discrete Neural Computation – A theoretical Foundation. Prentice Hall, 1995.

[23] I. Wegener. The Complwexity of Boolean functions. Wiley and Sons. Inc., 1987.

[24] M., Caudill. AI Expert: Neural Network Primer. Miller Freeman Publications 1990.

[25] R. E., Uhrig. Introduction to Artificial Neural Networks. Proceedings of the 1995 IEEE IECON 21st International Conference on Industrial Electronics, Control and Instrumentation, 1995, Vol. 1, Nov. 1995.

[26] K. Yale. Preparing the right data for training neural networks. IEEE Spectrum, Vol. 34, Issue 3, pp. 64-66, Mar. 1997.

[27] G. Stegmayer and O. Chiotti. The Volterra representation of an electronic device using the Netural Network parameters. Latin American Conference on Informatics (CLEI'2004), Sep. 2004.

[28] http://www.eco.utexas.edu/faculty/Kendrick/frontpg/NeuralNets.htm

[29] F. Somenzi. CUDD: CU Decision Diagram Package. ftp://vlsi.colorado.edu/ pub/., 2003.

[30] J. Lawrence. Introduction to Neural Networks – Design, Theory and Applications. California Scientific Software Press, 1994.

[31] T. Masters. Signal and Image Processing with Neural Networks. John Wiley & Sons, Inc., 1994.