Journal of Engineering Science and Technology
Vol. 11, No. 2 (2016) 177 - 192

SYNTHESIS AND REDUCED LOGIC GATE REALIZATION OF MULTI-VALUED LOGIC FUNCTIONS USING NEURAL NETWORK DEPLOYMENT ALGORITHM

A. K. CHOWDHURY[1], A. K. SINGH[2]

[1]School of Computer, University College of Technology Sarawak
Persiaran Brook, 96000 Sibu, Sarawak, Malaysia
[2]Department of Computer Application, National Institute of Technology, Kurukshetra, India
*Corresponding Author: adibkabir@ucts.com.my

Abstract

In this paper an evolutionary technique for synthesizing Multi-Valued Logic (MVL) functions using Neural Network Deployment Algorithm (NNDA) is presented. The algorithm is combined with back-propagation learning capability and neural MVL operators. This research article is done to observe the anomalistic characteristics of MVL neural operators and their role in synthesis. The advantages of NNDA-MVL algorithm is demonstrated with realization of synthesized many valued functions with lesser MVL operators. The characteristic feature set consists of MVL gate count, network link count, network propagation delay and accuracy achieved in training. In brief, this paper depicts an effort of reduced network size for synthesized MVL functions. Trained MVL operators improve the basic architecture by reducing MIN gate and interlink connection by 52.94% and 23.38% respectively.

Keywords: Multi-valued logic, Machine learning, Back-propagation, Training, Synthesis, Realization, Neural network

1. Introduction

In the recent years, Multi-Valued Logic (MVL) synthesis and simplification of logic functions has become a major research field of study. MVL is defined as a non-binary logic and involves the switching between more than two states [1]. MVL considerably reduces the number of lines required for parallel transmission of large data compared to binary counterpart [2]. Most of the effective MVL synthesis algorithms do not expedite in learning capability. The lack of any learning

Nomenclatures

$M_{constant}$	Momentum constant
N_{Output}	Neuron output
R	Set of real numbers
v_b	Level base voltage
v_{ib}	Initial base voltage value
$\overline{w}_n$	Weight vector
${}^{a}X_{n}^{b}$	Window literal
$\overline{x}_n$	Column vector
y_j^H	Output of hidden layer
y_j^O	Output from output layer
z	Variable length

Greek Symbols

β_{hidden}	Total hidden neurons from hidden layer
β_{input}	Total input to 1st layer
β_j^H	Inputs coming from input layer
β_j^O	Input from hidden layer
β_{output}	Total output neurons from output layer
$\Delta\omega$	Weight change
$\Delta\omega_{previous}$	Previous weight change
φ	Transfer function
$\varphi(v)$	Hyperbolic tangent function
ω	Weights of neural network
ω_{ij}^H	Weights from input to hidden neurons
ω_{ij}^O	Weights from hidden to output neurons

algorithm for MVL networks is an important restraint to many applications. In the process of learning, adaption also provides a degree of robustness by compensating for minor variability. Multi valued neuron modelling and realization of MVL functions using Neural Network (NN) is observed in [3-4]. NN approach is efficient in terms of accuracy, but lacks in memory consumption. It has been observed that neural network synthesis takes more hardware in terms of neuron count and interconnections [5]. Therefore, in the last decade, researchers are more focused on evolutionary algorithms. Thus an effort of reducing neuron count in neural network MVL synthesis is undertaken in this article.

In this paper, we report a learning MVL network construction, its algebraic properties and algorithm. Neural Network Deployment Algorithm (NNDA) is presented to show the synthesis and realization process of MVL functions through neural net. The algorithm is combined with feed-forward back-propagation learning capability and novel MVL operators from [6]. Firstly an elaborative description of MVL operators is presented. Afterward, a basic construction platform for NNDA is created which serves as the basis for realizing all the MVL

operators. The operators are further trained and tested using NNDA for realizing logic gates. Separate module for each gate is constructed. Each neural net logic operator is used for the synthesis of the logic expression. Next, simulation results and analysis shows that the number of neuron count in the quaternary function network reduces remarkably using NNDA-MVL algorithm. Subsequently, comparison and discussion of the result is presented to suggest that the proposed MVL synthesis method observes behavior of neural operators to achieve efficient synthesis. Finally, a conclusion is drawn.

2. Preliminaries

In this section a detailed description of the MVL operators are represented. Operators are used to train the neural network. The construction of the network architecture is presented as a basis for all the neural operators.

2.1. Description of MVL logic operators and algebra

Consider an *m*-valued *y*-variable function f(x), where $x = \{x_1, x_2, \text{L}\ , x_m\}$ and x_i takes on values from $R = \{0, 1, 2, \text{L}\ , y-1\}$, where "*y*" is the radix. The function f(x) is a mapping $f : R^{\,y} \Rightarrow R$. There are y^{y^m} different possible functions [7-10]. According to [6] few novel logic operators such as, ODD, EVEN, EXTENDED AND, WINDOW LITERAL are represented in this section of the paper. WINDOW LITERAL is represented with a new definition.

Definition 2.11- A MIN operator is defined as below, where a_n and b_n are definite values and both of them are a member of set of all real values *R* [1-2, 5-7] such that $a_1, a_2, \text{L}, a_n \in R$, $b_1, b_2, \text{L}, b_n \in R$ and $0 \le \{a, b\} \le (y-1)$. Therefore $MIN(a,b) = a \wedge b = a \cap b$. The MIN operator is represented by the following [7-10]:

$$MIN(a,b) = \begin{cases} a & if \quad a < b \\ b & otherwise \end{cases} \tag{1}$$

Definition 2.12- A MAX operator is defined as below, where a_n and b_n are definite values and both of them are a member of set of all real values *R* [7-9] such that $a_1, a_2, \text{L}\ , a_n \in R$, $b_1, b_2, \text{L}\ , b_n \in R$ and $0 \le \{a, b\} \le (y-1)$. Therefore $MAX(a,b) = a + b = a \cup b$. MAX operator is represented as following:

$$MAX(a,b) = \begin{cases} a & if \quad a > b \\ b & otherwise \end{cases} \tag{2}$$

Definition 2.13- The COMPLEMENT of *x* is defined as below [9-11] where $x \in R$, $a_1, a_2, \text{L}\ , a_n \in R$, $b_1, b_2, \text{L}\ , b_n \in R$, $0 \le \{a, b\} \le (y-1)$ and $a \le x \le b$. Therefore complement of *x* is represented as:

$$\overline{x} = y - 1 - x \tag{3}$$

Definition 2.14- The WINDOW LITERAL or SHORT LITERAL is defined as:

$$ {}^{a}x^{b} = \begin{cases} b & if \quad x = a \\ 0 & otherwise \end{cases} \tag{4} $$

where $a_1, a_2, \text{L}\ , a_n \in R$, $b_1, b_2, \text{L}\ , b_n \in R$ and $0 \le \{a, b\} \le (y-1)$

Definition 2.15- An EVEN operator is defined as below where m is defined as a standard value for detecting an even value, where $m = 2$. m is a member of set of all real values R and $a_1, a_2, \text{L}\ , a_n \in R$, $b_1, b_2, \text{L}\ , b_n \in R$, $\{p, m\} \in R$ and $0 \le \{a, b\} \le (y-1)$. The even operator detects whether value p is even or not. If p is even then $EVEN(p) = true$ and $EVEN(p) = p$, otherwise $EVEN(p) = false$ and $EVEN(p) = 0$ .Therefore the operation is presented as below:

$$ EVEN(p) = \begin{cases} p & if \quad p \bmod m = 0 \\ 0 & otherwise \end{cases} \tag{5} $$

And the even function is presented by the following relation:

$$ EVEN\_Func(a,b) = \begin{cases} \max(even(a), even(b)) & if \quad even(a) \,||\, even(b) = true \\ 0 & otherwise \end{cases} \tag{6} $$

Definition 2.16- An ODD operator is defined as below where a_n, m, k are definite values and m is defined as a standard value for detecting an odd value, where m is 2. k is defined as a standard value for detecting an odd value, where $k = 1, \ , m$ and k is a member of set of all real values R where $a_1, a_2, \text{L}\ , a_n \in R$, $b_1, b_2, \text{L}\ , b_n \in R$, $\{p, m, k\} \in R$ and $0 \le \{a, b\} \le (y-1)$. The ODD operator detects whether value p is odd or not. If p is odd then $ODD(p) = true$ and $ODD(p) = p$ otherwise $ODD(p) = false$ and $ODD(p) = 0$. Therefore the operation is defined as below:

$$ ODD(p) = \begin{cases} p & if \quad (p-k) \ge 0 \quad and \quad (p-k) \bmod m = 0 \\ 0 & otherwise \end{cases} \tag{7} $$

The ODD function is represented as below:

$$ ODD\_Func(a,b) = \begin{cases} \max(odd(a), odd(b)) & if \quad odd(a) \,||\, odd(b) = true \\ 0 & otherwise \end{cases} \tag{8} $$

For any odd input the output will be odd. If both odd input is detected the output prompts to zero.

Definition 2.17- An EXTENDEDAND operator is defined as below:

$$ {}^{a}\Gamma_{x_1}^{b_1, b_2} \bullet x_2 = \begin{cases} b_1 & if \quad (x_1, x_2) = (a, b_1) \\ b_2 & if \quad (x_1, x_2) = (a, b_2) \\ 0 & otherwise \end{cases} \tag{9} $$

The EXTENDED AND operator triggers a non-zero output only when $x_1 = a$ and $x_2 = b_1 \square b_2$, where $a_1, a_2, \mathrm{L}\ , a_n \in R$, $b_1, b_2, \mathrm{L}\ , b_n \in R$ and $0 \le \{a, b, x_1, x_2\} \le (y-1)$. The value of EXTENDED AND operator is determined by x_2. Considering the definition of the operator, it provides an opportunity for eliminating the usage of MIN operator twice in one expression. Therefore, in this paper, EXTENDED AND is extensively used during the synthesis of MVL functions. For example, if there exist an implicant of ${}^{0}x_1{}^{1}\mathrm{g}^{1}x_2{}^{1} + {}^{0}x_1{}^{2}\mathrm{g}^{2}x_2{}^{2}$, the implicant can be simplified using MVL algebraic rules and EXTENDED AND. Finally the simplified implicant can be represented as ${}^{0}\underline{x}_1^{1,2}({}^{1}x_2{}^{1} + {}^{2}x_2{}^{2})$.

Definition 2.18- The MVL matrix ${}^{a}x_1{}^{b}$ and ${}^{a}x_2{}^{b}$ represents the value of a minterm. Both of them are WINDOW LITERALs. The logic helps to form the representation of minterms with the coordinates (x_1 , x_2). An MVL function f_1 is shown as [012201120]. Considering coordinate $(x_1, x_2) = (2,1) = 2$, the value of the minterm is 2. When minterm is 2 then b = 2; $x_1 = 2$ therefore $a_1 = 2$ and $x_2 = 1$ therefore $a_2 = 1$. Thus ${}^{2}x_1{}^{2}\mathrm{g}^{1}x_2{}^{2}$ representation could be achieved. According to the definition of window literal, the logic expression could be presented as ${}^{2}x_1{}^{2} \bullet {}^{1}x_2{}^{2} = 2 \bullet 2$ or 2. Therefore an ideal minterm is presented as MIN operation of two WINDOW LITERALs as ${}^{a_m}x_1{}^{b_n} \bullet {}^{a_m}x_2{}^{b_n}$, where $a_1, a_2, \mathrm{L}\ , a_m \in R$, $b_1, b_2, \mathrm{L}\ , b_n \in R$, $0 \le \{a, b, x_1, x_2\} \le (y-1)$, $a_m \ne a_m$ and $b_n = b_n$. The literal bounds are defined as below:

$$ {}^{a_1}x_1{}^{b_1} = \begin{cases} b_1 & if \quad x_1 = a_1, \quad where \quad b_1 = \min term \quad value \\ 0 & otherwise \end{cases} \tag{10} $$

$$ {}^{a_2}x_2{}^{b_2} = \begin{cases} b & if \quad x_2 = a_2, \quad where \quad b_2 = \min term \quad value \\ 0 & otherwise \end{cases} \tag{11} $$

2.2. Basic structure of NNDA architecture

The feed-forward back-propagation is applied to the multiple layers of the neurons for constructing NNDA neural architecture. This neural network is composed of three layers. First, second and third layers are consecutively called input, hidden and output layer. Depending on the MVL operator the number of inputs varies. The r[th] column of each input is represented as a column vector $\underline{\underline{x}}_n$ which varies between 0, 1, 2 and 3 values. The column vector consists of real entities where $x \in R, x \ge 0, 0 < n \le 5$ and $\underline{\underline{x}}_n \in \{x_0, x_1 \mathrm{K}\ x_4\}$. Column vector is presented as $\underline{\underline{x}}_n \in [x_0, x_1 \mathrm{K}\ x_4]^T$ [12-13].

For a five different input column vectors, the inputs are shown as $\underline{\underline{x}}_0 = [x_0, x_1 \mathrm{K}\, x_r]^T; \underline{\underline{x}}_1 = [x_0, x_1 \mathrm{K}\, x_r]^T; \underline{\underline{x}}_2 = [x_0, x_1 \mathrm{K}\, x_r]^T; \underline{\underline{x}}_3 = [x_0, x_1 \mathrm{K}\, x_r]^T \, and \, \underline{\underline{x}}_4 = [x_0, x_1 \mathrm{K}\, x_r]^T$. Each of the input vectors represent the input layer of the neural network. The weights are distributed evenly with each of the inputs. The r[th] column of each

synaptic weight is represented as a column weight vector $\vec{w}_n$ which varies between -1 to 1. The weight vector consists of real entities where $w \in R, -1 \le w \le 1, n \le r$ and $\vec{w}_n \in \{w_0, w_1 \mathrm{K}\ w_r\}$. The weight vector is represented as $\vec{w}_n \in [w_0, w_1 \mathrm{K}\ w_r]^T$. The corresponding weights for each of the inputs are multiplied with their respective inputs. The summation of the weights multiplied by the inputs is represented as $\sum_{i=0}^{r} w_i \mathrm{g} x_i = w_0 \mathrm{g} x_0, w_1 \mathrm{g} x_1 \mathrm{K}\ w_r \mathrm{g} x_r$ where $\vec{w}_n \mathrm{g} \vec{x}_n \in [w_0 \mathrm{g} x_0, w_1 \mathrm{g} x_1 \mathrm{K}\ w_r \mathrm{g} x_r]$ [13-14]. The neuron has a bias b which is scalar. The bias is added to the summation of weight and input to adjust the output from the neuron. Bias b is similar to weight, but bias and weight differs by a constant input of 1. The hyperbolic tangent sigmoid transfer function is represented as below:

$$\varphi(v) = \frac{2}{1+e^{-2 \cdot n}} - 1 = \frac{1-e^{-2 \cdot n}}{1+e^{-2 \cdot n}} = \frac{e^{2 \cdot n}-1}{e^{2 \cdot n}+1}; \quad [where \quad n = v] \tag{12}$$

The transfer function φ is different in functionalities depending on the desired output. The tangent sigmoid transfer function is presented by $\varphi(v)$, where $v = \vec{w}_n \mathrm{g} \vec{x}_n + b$. The neuron output is shown as [12-14].

$$N_{Output} = \varphi\left(\vec{w}_n \mathrm{g} \vec{x}_n + b_n\right) = \varphi\left(\left[\sum_{i=0}^{r} w_i \mathrm{g} x_i\right] + b_i\right) \tag{13}$$

The output layer neuron has the same hyperbolic tangent sigmoid transfer function. Assume the layers of the neural network is characterized by β. Therefore the total number of inputs in the input layer is identified by β_{input}. Similarly total number of hidden and output neurons from respective layers is observed by β_{hidden} and β_{output}. The weights of the neural network is characterized by ω, "*i*" represents the sequence of inputs and "*j*" represents the sequence of hidden neurons. Inputs coming from the input layer associated with respective weight and bias is represented by equation 14. Output of the hidden layer for respective hidden neurons shown in equation 15 [12-14].

$$\beta_j^H = \sum_{i=0}^{\beta_{input}} \left(\omega_{ij}^H \mathrm{g} x_i\right) + b_j \tag{14}$$

$$y_j^H = \varphi\left(\beta_j^H\right) \tag{15}$$

3. Proposed Methodology

The NNDA architecture construction process is shown in this section. The architecture serves as a basic platform for creating different module for different logic operators. A pseudo-code for NNDA algorithm is also presented. An elaborative discussion on NNDA synthesis is shown in later part of the section.

3.1. Neural network deployment algorithm (NNDA) in MVL function synthesis

The construction of the MVL neural net begins with the algorithm provided in Fig. 1. The algorithm initializes all the input, output and targets. Based on this procedure the NNDA algorithm performs further processing in realization. NNDA not only reduces the product term but also reduces the neural net block for representing each of the MVL functions. The algorithm determines all the non-zero minterm from 1st column to nth column. NNDA converts a MVL function into a matrix. Afterward, the algorithm finds each column and their respective rows for any non-zero minterms. Each column of the matrix represents one or more effective implicants of NNDA-MVL synthesis method. Each of the implicant is formed of different number of EXTENDED AND and MIN operators. MIN is trained by a neural network and NN_MIN_i is formed. The implicants are combined by operator MAX for every two columns.

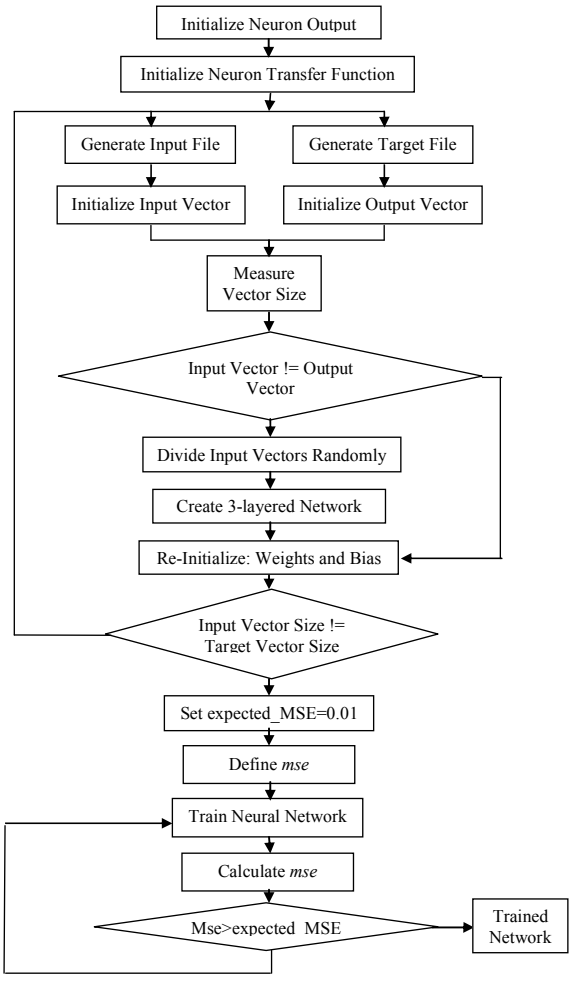


Fig. 1. The procedure which builds the MVL neural architecture.

Consecutively, MAX logic operator is universal, trained for synthesis and do not associate to the column sequence. Upon MAX neural training, NN_MAX_i is formed. Based on NNDA a pseudo code has been prepared. The entire pseudo code for the algorithm is represented in Fig. 2.

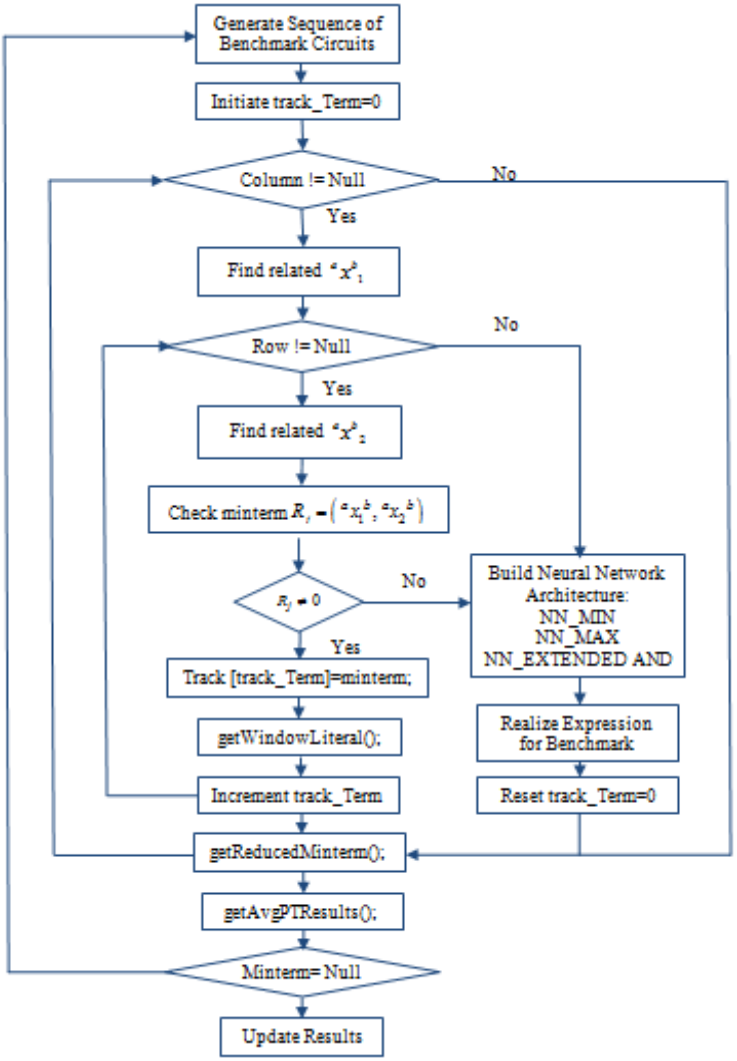


Fig. 2. NNDA-MVL pseudo code.

An example of an implicant is shown in [6]. The implicant is composed of ${}^{0}x_1^{1}\mathrm{g}^{1}x_2^{1} + {}^{0}x_1^{2}\mathrm{g}^{2}x_2^{2} + {}^{0}x_1^{3}\mathrm{g}^{3}x_2^{3}$ and extracted from a MVL function. Assume the implicant is derived from column one. Now the partial implicant can be represented as $I_a = {}^{0}\Gamma_1^{1,2}({}^{1}x_2^{1} + {}^{2}x_2^{2})$. The other half of the implicant can be expressed as $I_b = {}^{0}x_2^{3}\mathrm{g}^{3}x_2^{3}$. This expression is further simplified as

$I_a = {}^{0}\Gamma_{1}^{1,2}({}^{1}x_2^{1} + {}^{2}x_2^{2})$ using the EXTENDED AND operator. Therefore $I = I_a + I_b$ or $I = NN\_ExAND_0 + NN\_MIN_0$. Upon simplifying all the implicants, MAX operator operates on all the sub-implicants to prompt the final output for the 1st column. The same process repeats for 2nd and 3rd column and so on.

4. Simulation Analysis

In this section, MVL operations are investigated by NNDA algorithm. MVL operators are trained by the algorithm. It is observed that the neural network takes less than 300 iterations to fully converge to the behaviour of the logic operator. A sample of 5000 benchmark training sets is sequentially created to train each of the neural network system. A consistent neural architecture is used as basis for all logic operators.

4.1. EXTENDED AND neural network

The EXTENDED AND architecture consists of feed-forward back-propagation and five inputs. The inputs x_1, a, b_1, b_2 and x_2 are used to train the neural network. The network takes the inputs and predicts the possible outputs. The construction of neural network explains detail on the basic architecture of the network. The training, validation and test receiver operating characteristics (ROC) is observed by Figs. 3 and 4. A strong classification is realized when the output values are leant towards the left axis and closer to 1. ROC is used to check the quality of the classifiers.

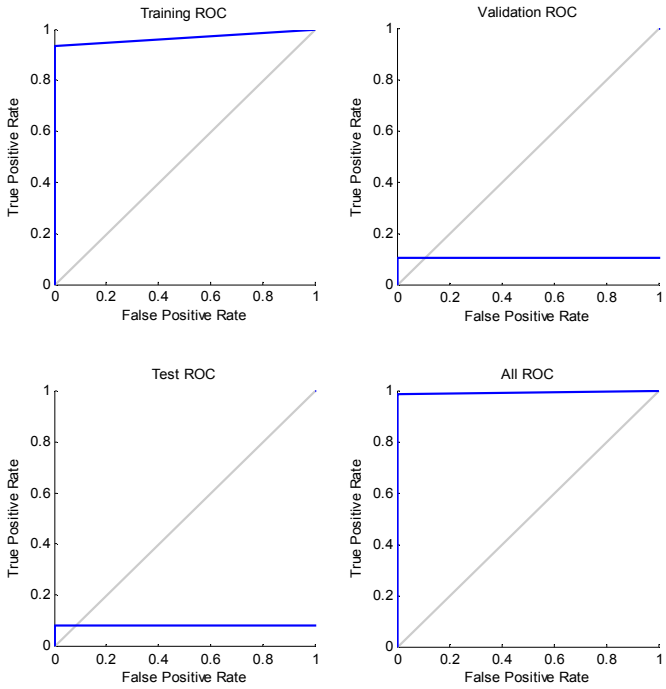


Fig. 3. ROC observation for EXTENDED AND.

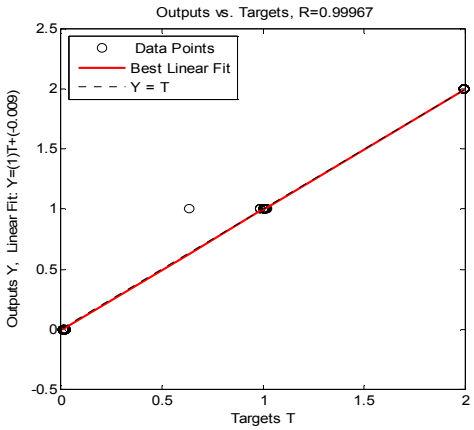


Fig. 4. Accuracy of EXTENDED AND operation.

The neural network trains itself until it fully converges to the behaviour of the logic operator. The operator is used as one of the basic component to realize and synthesize the MVL functions. The neural operator is trained with 3 valued 2 variable MVL functions.

It is observed that the performance reaches as low as up to 0.0001068 within 71 epochs. The best linear fit is observed at 99.97%. Among training benchmarks, a random quantity of data is chosen for testing and validation. Upon training an additional of 1K benchmark is used for testing the logic operator. Floating point variables are avoided for the training purpose. The inputs to the network are limited by 0-2 values. The testing performance and post training accuracy is shown in Fig. 3 and Fig. 4. Random weights are initialized with every input to the network. A total of 20 weights which are associated with inputs is observed from Fig. 5. It is realized that within 71 epochs the gradient reaches 0.000203 and 6 validation check is observed. The gradient analysis and the validation check is observed by the Fig. 6.

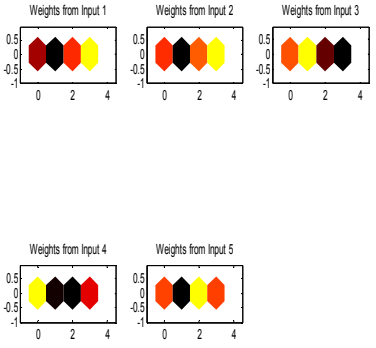


Fig. 5.Weight initialization for x_1, x_2, b_1, b_2 and *a*.

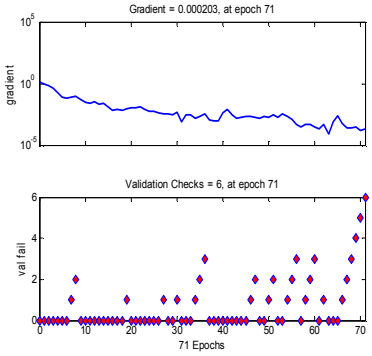


Fig. 6. Gradient, validation after 71 epochs.

4.2. ODD neural network

Basic neural ODD is trained with 4 valued 2 variable benchmarks. Among all benchmarks a random quantity of data has been chosen for testing and validation. The MSE reaches as low as up to 0.696 within 101 epochs. The best linear fit has been observed approximately 75.97%. An additional of 1K benchmarks is used for testing the logic operator after training. Floating point variables are considered for the training purpose for better linear data fit. The inputs to the network have been limited by 0, 1, 2 and 3. Weight initialization, post training accuracy and gradient analysis is observed from Figs. 7, 8 and 9. It is observed that within 107 epochs the gradient has reached 0.107 and 6 validation checks has commenced.

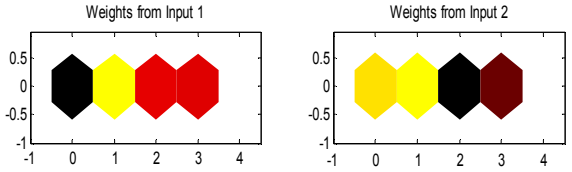


Fig. 7.Weight initialization for x_1 and x_2.

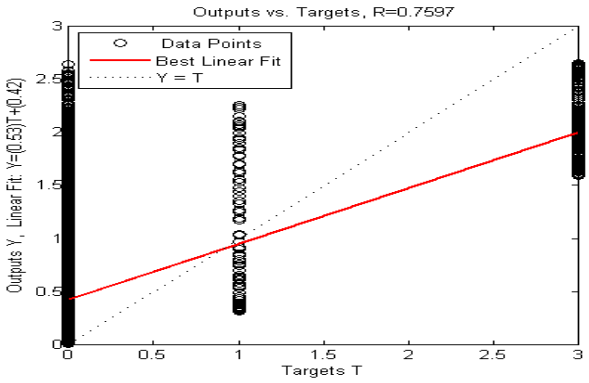


Fig. 8. Accuracy of ODD operation.

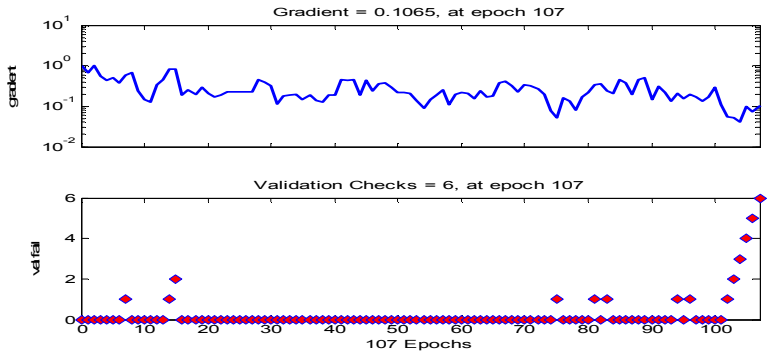


Fig. 9. Gradient, validation after 107 epochs.

4.3. EVEN neural network

MVL-EVEN is trained with 4 valued (0-3) benchmarks. Additional benchmark is used for post-training evaluation. Floating point variables are considered for the training purpose for better linear data fit. The best linear fit is observed at 97.38%. The *mse* reaches as low as up to 0.0488 within 129 epochs. A total of 8 weight initialization, testing performance and gradient analysis of the EVEN operator is observed by Figs. 10, 11 and 12.

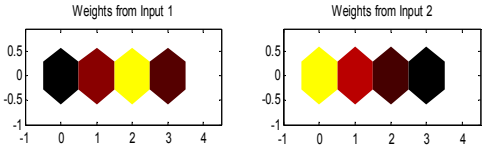


Fig. 10.Weight initialization for x_1 and x_2.

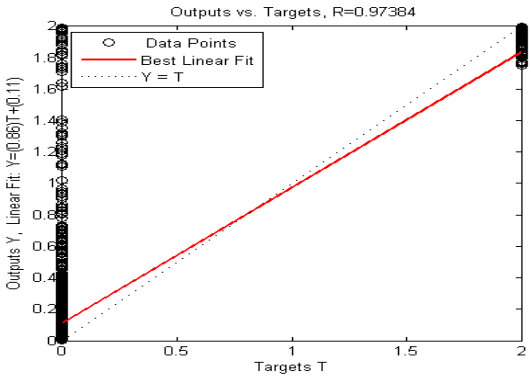


Fig. 11. Accuracy of EVEN operation.

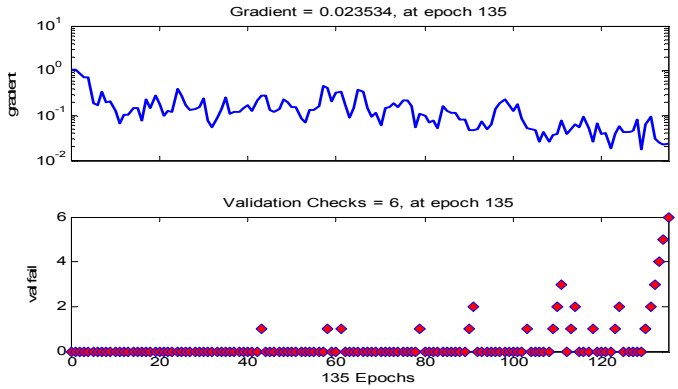
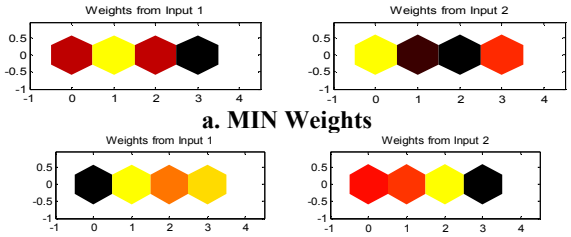


Fig. 12. Gradient, validation after 135 epochs.

4.4. MIN and MAX neural network

The MIN & MAX operator is trained with 5K 4-valued benchmarks. Additional 1K benchmark is used for testing the logic operators. Floating point variables are

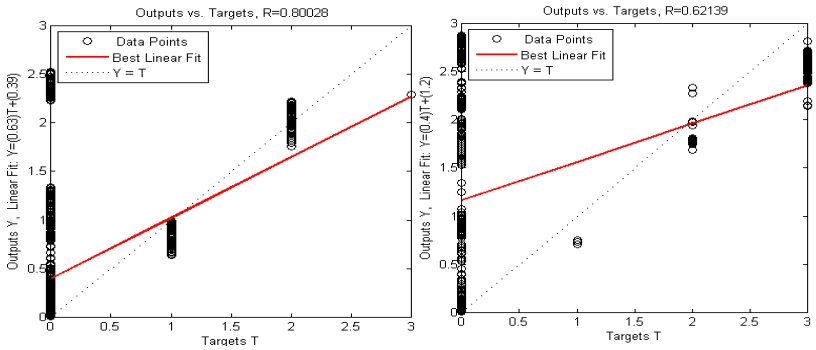
considered for better linear data fit. The performance is measured up to 0.292 for MIN and 0.867 for MAX consecutively within 105 and 66 epochs. MIN has a best linear fit of 80.03% compared to MAX. It is observed that MAX achieves a fit of 62.14% within 111 epochs while the gradient reaches 0.012. On the other hand MIN reaches 0.034 within 72 epochs . On both cases validation check is found to be 6. Weight initialization, testing performance and gradient analysis of the MIN and MAX operator is shown in Fig. 13(a,b), 14(a,b) and 15(a,b).



a. MIN Weights

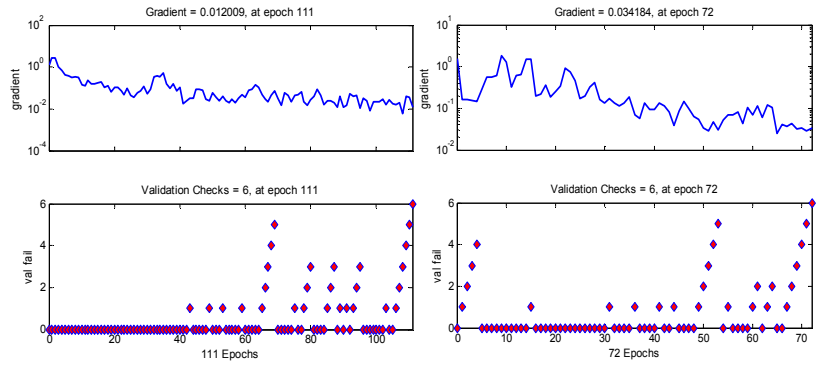
b. MAX Weights

Fig. 13. Weight initialization of MIN and MAX operator.



a. MIN Operator b. MAX Operator

Fig. 14. Accuracy of MIN and MAX operation.

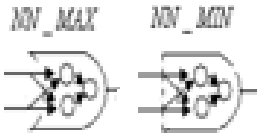


(a) Neural MIN **(b) Neural MAX**

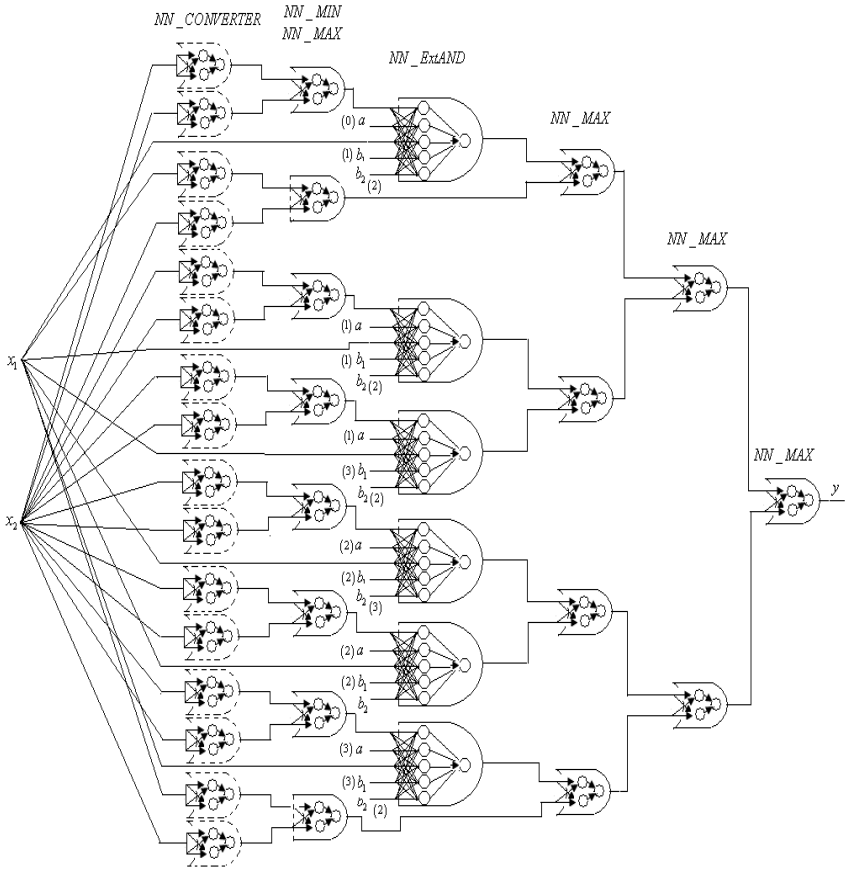
Fig. 15. Gradient and validation analysis for MIN and MAX neural operator.

5. Results and Discussion

MVL gate count and network link count is observed when different neural net logic operators are interconnected to realize synthesized MVL functions. Fig. 16(a, b) illustrates the entire reduced neural network for realizing the quaternary function in [5]. Table 1 shows NNDA-MVL comparison and reduction in logic operators and interconnected link in the neural network.



(a) Neural MIN and MAX symbol.



(b) Ternary function realization.

Fig. 16. Networks for a synthesized MVL function.

Table 1. Post training comparison among different MVL neural net operators.

	[5]	NNDA-MVL	Improved Reduction Achievement (%)
EXTENDED AND,			
MIN Operator	17	8	52.94%
MAX Operator	13	13	00.00%
Interconnected Links	77	59	23.38%

The reduced network incorporates lesser MVL neural operators for realizing a logic expression. The network also reduces the number of internal connections between logic operators. It is observed that some trained MVL network provides an efficient and fast output. A total of

6 NN_ExtAND, 2 NN_MIN, 13 NN_MAX and NN_CONVERTER

logic operators are used for the network architecture.

In this section, a comparative analysis is carried out considering different MVL neural network operators. CPU time taken to train, propagation delay, number of gates, hidden neurons and accuracy are mainly considered during analysis. The INVERTER operator requires the least time 8.52 seconds to train. It also exhibits the least transition delay of 0.0084 seconds. The highest accuracy of 99.97% is observed by EXTENDED AND neural operator. Among MIN, MAX, EVEN and ODD operators, ODD not only requires the least amount of time 9.09 seconds to train but also exhibits the least propagation delay of 0.010 seconds. EVEN achieved the highest accuracy of 97.38%.

6. Conclusion

In this paper we have discussed a set of basic MVL operators. Hybrid combination of MVL operators and NNDA reduces the network size in terms of logic gate and internal links. The obtained are compared using a benchmark neural network which realizes the same function in [5]. Post-test evaluation, weight initialization, gradient and validation analysis is done to observe the anomalistic characteristic of the neural MVL operators. The results achieved have shown that the NNDA-MVL algorithm outperforms the existing neural network realization. The result depicts an improvement of 52.94% for MVL MIN gate reduction and 23.38% internal link reduction for a given MVL network.

References

1. Lablan, P. (2005-2006). Multi-valued logic. Retrieved July 6, 2014, http:// archive.is/rIyG.
2. Smith, K.C. (1998). A multiple valued logic: a tutorial and appreciation. *IEEE Transactions on Computers*, 21(4), 17-27.
3. Tang, C.; Chen, F.; and Li, X. (2011). Perceptron implementation of triple-valued logic operations. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(9), 590 – 594.
4. Peng, D.; Feng D.; and Yingjiang, Li. (2010). The risk assessment of bank based on multiple correlation analysis and three-valued logic neuron model.

Proceedings of the Asia-Pacific Conference on Wearable Computing Systems (APWCS). Shenzhen, China, 154-160.

5. Zheng, T.; Qi-Ping, C.; and Ishizuka, O. (1998). A learning multiple-valued logic network: algebra, algorithm, and applications. *Journal of IEEE Transactions on Computers*, 47(2), 247-251.
6. Chowdhury A.K.; Raj N.; and Singh, A.K. (2013). A novel high deduction algorithm for synthesizing multiple-valued logic and circuit realization. *Journal of Multi-Valued Logic and Soft Computing*, (Under Review).
7. Sarica, F.; and Morgul, A. (2011). Basic circuits for multi-valued sequential logic. *Proceedings of the 7th International Conference on Electrical and Electronics Engineering*. Bursa, Turkey, II-66-II-68.
8. Jain, A. K.; Bolton, R. J.; and Abd-El-Barr, M. H. (1993). CMOS multiple-valued logic design part II: Function realization. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*. 40(8), 515-522.
9. Jain, A. K.; Bolton, R. J.; and Abd-El-Barr, M. H. (1993). CMOS multiple-valued logic design part I: Circuit implementation. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(8), 503-514.
10. Gawande, A.D.; and Ladhake, S. A. (2008). Constraints in the design of CMOS MVL circuits. *Proceedings of the 7th WSEAS International Conference on Instrumentation, Measurement, Circuits and Systems*. Hangzhou, China, 108-113.
11. Sarif, B. A. B.; and Abd-El-Barr, M. (2006). Synthesis of MVL functions - Part I: The genetic algorithm approach. *Proceedings of the International Conference on Microelectronics*. Dhahran, Saudi Arabia, 154-157.
12. Matsumoto, M.; Ueda, Y.; and Nomoto, I. (2000). The synthesis of multiple-valued logic circuits using local-excitation-type neuron models. *Proceedings of the 30th IEEE International Symposium on ISMVL*. Oregon, U.S.A, 21-26.
13. Ngom, A.; and Simovici, D.A. (2004). Evolutionary strategy for learning multiple-valued logic functions. *Proceedings of the 34th International Symposium on ISMVL*. Toronto, Canada, 154-160.
14. Zheng, T.; Ishizuka, O.; and Tanno, K. (1995). Learning multiple-valued logic networks based on backpropagation. *Proceedings of the 25th International Symposium on ISMVL*, Indiana, USA, 270-275.