

Low Power Neural Network Training Using A GMDH Type Algorithm

Ankur Agarwal, Assistant Professor, Florida Atlantic University, ankur@cse.fau.edu
A. S. Pandya, Professor, Florida Atlantic University, abhi@cse.fau.edu
Morrison S. Obeng, Professor, Bethune-Cookman University, obengm@cookman.edu

Abstract

Authors have developed a Group Method of Data Handling (GMDH) type algorithm for designing multi-layered neural networks. The algorithm is general enough that it will accept any number of inputs and any sized training set. Each neuron of the resulting network is a function of two of the inputs to the layer. The equation for each of the neurons is a quadratic polynomial. Several forms of the equation are automatically tested for each neuron to make sure that only the best equation of two inputs is kept. All possible combinations of two inputs to each layer are also tested by the algorithm and only the best neurons at each level are retained. The algorithm's goal is to create as accurate network as possible while minimizing the size of the network. Software was developed to train and simulate networks using our algorithm. Several applications were modeled using our software, and the results demonstrate that our algorithm succeeded in developing small, accurate and multi-layer networks. This paper further presents the hardware implementation of the algorithm that consumes less power than other such systems.

Keywords: Neural Networks, Group Method of Data Handling, GMDH, Simulations, Modeling, Pass Transistor Logic.

Introduction

Design of intelligent systems that can learn from the environment and adapt to the change in the environment has been pursued by many researchers in this age of information technology and has produced three types of Intelligent Control Systems. Type-1 comprises of a single expert system, which processes symbolic information and provides assistance to control engineers in decision making process for design and off-line monitoring. Type-2 consists of coupled systems that couple programs for numerical computation into an expert system such that it can be used to solve engineering problems. The last category of such systems includes Integrated Intelligent Systems that are large environments, which can integrate different expert systems or numerical packages together to solve complex problems. Neural networks, which are the circuitry for human intelligence, have been one of the prime candidates in this regard. An artificial neural network is a system consisting of small processing units (called neurons) that perform specific tasks in parallel. Despite many years of studies involving neural networks, it is only due to advancement in programmable hardware that actual implementation for study and their applications have become practical. Though there are many examples of neural network codes running on von Neumann computers, there are still few commercially available neural networks implemented in hardware [1] [2].

This paper introduces a hardware implementable algorithm that will design multi-layered neural networks to simulate a desired system with any number of inputs and any sized training set. Our algorithm uses concepts from Group Method of Data Handling (GMDH). Nodes in the hidden layers are developed for each layer that represents functions of every possible combination of two inputs to that layer. Unlike the traditional approaches for Neural Network design which result in a black box architecture for the intelligent system, our approach provides an optimal architecture for the network and provides an intelligent system that clearly spells out the functional relationships between inputs and outputs. Each node will consist of a function of up to six possible terms: A constant, the inputs (for linear input/output relationships), the square of the inputs (to allow for nonlinear (parabolic) input/output relationships), and the product of the

two inputs. The combination of these terms that gives the lowest Mean Squared Error (MSE) is kept as the transfer function of the neuron. The algorithm develops neurons for all possible combinations of two inputs to the layer. It then continues to choose only those neurons that supply the best possible MSE. The surviving neurons are then passed on to a new layer of the network. This continues until added layers no longer improve the MSE, the network has reached a designer defined layer limit, or there is only a single surviving node in a hidden layer. We demonstrate the ability of our approach by applying it to various real systems such as medical image recognition, system identification etc.

Another aspect of this design is the low power circuitry we are using for its implementation. The new design of algorithm has been employed by using complementary pass transistors. By looking into aspects like power consumption, cost effectiveness, speed and wiring complexity we are analyzed for the new design. The interesting fact in design of proposed algorithm is the design of the arithmetic operations, which shows a significant reduction in the number of nMOS and pMOS transistors employed in the design. This results in increasing the speed and a significant reduction in power consumption and layout complexity.

Neural Networks

An artificial neural network is a system consisting of small processing units (called neurons) that perform specific tasks in parallel. The neurons are arranged in layers, with the output of the neurons in a layer becoming the input to the neurons in the next. Typically the first layer consists of a neuron for each input to the system. These neurons simply hold the input values and pass them on to the next layer. The next layer is called a hidden layer since the user of the network does not have access to these neurons. The neurons themselves consists of a set of inputs from the first (input) layer, an implementation of a mathematical function of those inputs, then an output that carries the result of the function to the next layer. This next layer can potentially be another layer whose neurons are now functions of the outputs of the previous layer. After the final hidden layer, a last layer, called the output layer, actually supplies the predicted output values to the user. Some neural networks are capable of designing themselves. These networks are generally called self-organizing, and the process of setting up a network to perform a specific task is called training [3]. The mathematical functions in the hidden neurons are where the training actually occurs in the networks. Each input must be tested to see its effects on the output of the system. A comparison between the input and output results in a weight or coefficient that is associated with that input value.

One of the most common training methods is called back-propagation. In this method, an initial set of weights is assigned to each input. Then each input set from the training data is supplied to the network. The corresponding output is tested and compared to the expected output. A function of the error is used to update the coefficients. After testing all of the input sets from the training data, the network retests all of the data. This continues until the network determines that the coefficients are supplying the best output possible [3] and it does not generate the optimal network architecture. This method can potentially require millions of iterations before the final transfer functions are chosen. As we will discuss later in this paper, we have chosen a fast, statistically based method for finding the transfer function coefficients.

Group Method of Data Handling

The GMDH algorithm was first presented by Ukrainian engineer and cyberneticist, A.C. Ivakhnenko, and his colleagues in 1968 [4]. His intent was to develop a rival method to stochastic approximation. Originally designed to estimate higher order regression polynomials, this heuristic self-organization method has been applied to a large variety of fields including medicine, biology, manufacturing, environmental and ecological systems, psychology, economics, etc. The method builds hierarchical polynomial regression networks to model complex input-output relationships [5].

The GMDH algorithm presented in this paper generates and tests all input-output combinations for a system. Each element of the system implements a function of two inputs. Coefficients of the elements are determined using a regression technique. A threshold is specified at each level to determine if the outputs of the elements in a layer are giving acceptable results. If the result from an element is within the

threshold, it is passed on to the next layer. Those elements and variables that are least useful in predicting the proper output are filtered out. Each succeeding layer has more complex combinations. Layers are added until satisfactory results are reached [6, 7, 8]. It is almost a “Darwin” model: Only those elements that are strong and give desired results are allowed to pass on to the next stage. Using this method, the algorithm chooses the optimal set of input variables, the degree of nonlinearity in the final model, and the structure and the degree of interaction terms in the final model [9]. There are 4 advantages to this method: (1) A small training set is required; (2) The multiple layer structure of the resulting system results in a feasible way of implementing a high degree multinomial; (3) The computational burden is reduced; (4) Inputs/functions of inputs that have little impact on the output are automatically filtered out.

As with most neural networks, the first layer of our networks would be comprised of a single neuron per input to the system. These neurons’ sole functions are to pass the input in to the first hidden layer. The hidden layers would have the actual transfer functions that will attempt to predict the proper output. Since there are many different types of relationships between inputs and the output of a system, the first step to developing a GMDH based network is to narrow down which type of relationship the network will explore. GMDH algorithms often combine linear and non-linear elements in order to better find the input-output relationship of a system. In our algorithm, linear and parabolic (input squared) relationships are chosen. In addition, each neuron will have two inputs. This results in 6 possible terms. The following quadratic polynomial is used as the output equation for the hidden neurons:

$$Y = b_0 + b_1 in_1 + b_2 in_2 + b_3 in_1^2 + b_4 in_2^2 + b_5 in_1 in_2 \quad (1)$$

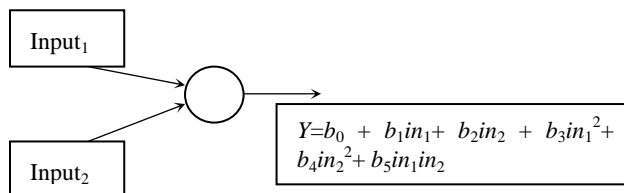


Figure 1. Diagram of Basic Neuron

In (1), Y is the output of the neuron, b_n are the coefficients of each term, and in_1 and in_2 are the two inputs selected for the neuron. The b_0 coefficient is added as a constant offset for the function. It can be used to offset noise or error in the system. Fig. 1 illustrates a sample hidden neuron. Fig. 2 shows the entire network architecture.

Training the Network

One of the most important aspects of a network is not necessarily how it calculates its outputs, but rather how it chooses which neurons and layers are placed in the final network. The size, speed, and accuracy of network are greatly impacted by this part of any training algorithm. The following is our method of calculating the coefficients of the individual neurons, a description of how the final set of neurons is chosen for a layer, and how layers are chosen for the final network.

Coefficients

Calculation of the coefficients for the neuron is probably the most complicated part of the entire design procedure. The following method is suggested in [10]. The training set supplied by the user will be used in the form of matrices. The algorithm systematically takes sets of two inputs to form the terms in (1). The result is the following matrix equation (2). In this case, s is the number of samples, $i_{m,n}$ represents input n at sample m , and y_m is the desired output for the input sample m . In terms of the inputs to the neuron, $i_1=1$, $i_2=in_1$, $i_3=in_2$, $i_4=in_1^2$, $i_5=in_2^2$ and $i_6=in_1 in_2$. The next step is to multiply both sides of the equation by the transpose of the input matrix. Multiplying both sides of (3) with the inverse of the 6X6 input matrix will give the six b coefficients. If there is no inverse to the input matrix, the equation does not converge (there is

not enough information or not enough variety of information to properly calculate the values for the given combination of terms).

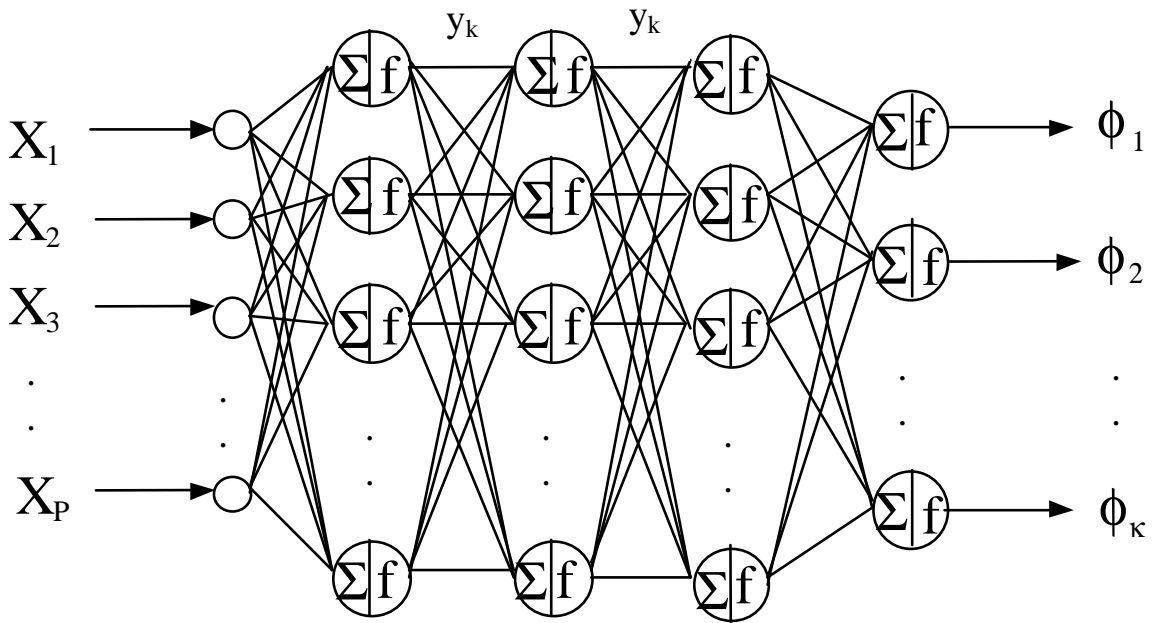


Figure 2. Network architecture for GMDH type neural networks

$$\begin{pmatrix} i_{1,1} & i_{1,2} & i_{1,3} & i_{1,4} & i_{1,5} & i_{1,6} \\ i_{2,1} & i_{2,2} & i_{2,3} & i_{2,4} & i_{2,5} & i_{2,6} \\ i_{3,1} & i_{3,2} & i_{3,3} & i_{3,4} & i_{3,5} & i_{3,6} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ I_{s,1} & i_{s,2} & i_{s,3} & i_{s,4} & i_{s,5} & i_{s,6} \end{pmatrix} \begin{pmatrix} b_0 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_s \end{pmatrix} = \begin{pmatrix} y_0 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_s \end{pmatrix} \quad (2)$$

$$\begin{bmatrix}
\sum_{j=1}^s i_{j,1}^2 & \sum_{j=1}^s i_{j,1}i_{j,2} & \sum_{j=1}^s i_{j,1}i_{j,3} & \sum_{j=1}^s i_{j,1}i_{j,4} & \sum_{j=1}^s i_{j,1}i_{j,5} & \sum_{j=1}^s i_{j,1}i_{j,6} \\
\sum_{j=1}^s i_{j,1}i_{j,2} & \sum_{j=1}^s i_{j,2}^2 & \sum_{j=1}^s i_{j,2}i_{j,3} & \sum_{j=1}^s i_{j,2}i_{j,4} & \sum_{j=1}^s i_{j,2}i_{j,5} & \sum_{j=1}^s i_{j,2}i_{j,6} \\
\sum_{j=1}^s i_{j,1}i_{j,3} & \sum_{j=1}^s i_{j,2}i_{j,3} & \sum_{j=1}^s i_{j,3}^2 & \sum_{j=1}^s i_{j,3}i_{j,4} & \sum_{j=1}^s i_{j,3}i_{j,5} & \sum_{j=1}^s i_{j,3}i_{j,6} \\
\sum_{j=1}^s i_{j,1}i_{j,4} & \sum_{j=1}^s i_{j,2}i_{j,4} & \sum_{j=1}^s i_{j,3}i_{j,4} & \sum_{j=1}^s i_{j,4}^2 & \sum_{j=1}^s i_{j,4}i_{j,5} & \sum_{j=1}^s i_{j,4}i_{j,6} \\
\sum_{j=1}^s i_{j,1}i_{j,5} & \sum_{j=1}^s i_{j,2}i_{j,5} & \sum_{j=1}^s i_{j,3}i_{j,5} & \sum_{j=1}^s i_{j,4}i_{j,5} & \sum_{j=1}^s i_{j,5}^2 & \sum_{j=1}^s i_{j,5}i_{j,6} \\
\sum_{j=1}^s i_{j,1}i_{j,6} & \sum_{j=1}^s i_{j,2}i_{j,6} & \sum_{j=1}^s i_{j,3}i_{j,6} & \sum_{j=1}^s i_{j,4}i_{j,6} & \sum_{j=1}^s i_{j,5}i_{j,6} & \sum_{j=1}^s i_{j,6}^2
\end{bmatrix}
\begin{bmatrix}
b_0 \\
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5
\end{bmatrix}
=
\begin{bmatrix}
\sum_{j=1}^s i_{j,1} y_j \\
\sum_{j=1}^s i_{j,2} y_j \\
\sum_{j=1}^s i_{j,3} y_j \\
\sum_{j=1}^s i_{j,4} y_j \\
\sum_{j=1}^s i_{j,5} y_j \\
\sum_{j=1}^s i_{j,6} y_j
\end{bmatrix} \quad (3)$$

The Best Equation for a Neuron

One of the benefits of only taking two inputs into a neuron is that it increases the chances of finding a close relationship between an input and the output. Including all six terms may actually hinder the neurons ability to find this relationship. Therefore, instead of automatically accepting the equation with all six terms, the network software tests all of the possible combinations of the terms. All possible combinations of each of the terms are removed and the coefficients are recalculated. The number of subsets (S) containing E elements that can be made from a set of N elements can be derived the following

$$S = \frac{N!}{(N - E)!E!} \quad (4)$$

Since there are a total of six terms, $N=6$. Using the equation, there are six single term equations ($E=1$). For example, $y=b_0$, $y= b_1in_1$, and $y= b_2in_2$. Similarly, there are 13 two-term equations ($E=2$), 20 three-term equations ($E=3$), 20 four-term equations, and 15 five-term equations ($E=4$). With the one six-term equation already discussed, there are 77 possible equations for each neuron.

New b coefficients are calculated for each of these equations. To do so, (3) can still be used. The difference will be that the proper rows and columns must be removed from input matrix, and the proper row from the output matrix. For example, to recalculate the b coefficients without the in_1 term, row 2 and column 2 of the 6x6 input matrix are removed (remember that i_1 is the constant term and i_2 represents in_1). Similarly, row 2 of the output matrix is removed. Therefore, (3) can be re-written as:

$$\begin{bmatrix} \sum_{j=1}^s i_{j,1}^2 & \cdot & \sum_{j=1}^s i_{j,1} i_{j,3} & \sum_{j=1}^s i_{j,1} i_{j,4} & \sum_{j=1}^s i_{j,1} i_{j,5} & \sum_{j=1}^s i_{j,1} i_{j,6} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \sum_{j=1}^s i_{j,1} i_{j,3} & \cdot & \sum_{j=1}^s i_{j,3}^2 & \sum_{j=1}^s i_{j,3} i_{j,4} & \sum_{j=1}^s i_{j,3} i_{j,5} & \sum_{j=1}^s i_{j,3} i_{j,6} \\ \sum_{j=1}^s i_{j,1} i_{j,4} & \cdot & \sum_{j=1}^s i_{j,3} i_{j,4} & \sum_{j=1}^s i_{j,4}^2 & \sum_{j=1}^s i_{j,4} i_{j,5} & \sum_{j=1}^s i_{j,4} i_{j,6} \\ \sum_{j=1}^s i_{j,1} i_{j,5} & \cdot & \sum_{j=1}^s i_{j,3} i_{j,5} & \sum_{j=1}^s i_{j,4} i_{j,5} & \sum_{j=1}^s i_{j,5}^2 & \sum_{j=1}^s i_{j,5} i_{j,6} \\ \sum_{j=1}^s i_{j,1} i_{j,6} & \cdot & \sum_{j=1}^s i_{j,3} i_{j,6} & \sum_{j=1}^s i_{j,4} i_{j,6} & \sum_{j=1}^s i_{j,5} i_{j,6} & \sum_{j=1}^s i_{j,6}^2 \end{bmatrix} \begin{bmatrix} b_0 \\ \cdot \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^s i_{j,1} y_j \\ \cdot \\ \sum_{j=1}^s i_{j,3} y_j \\ \sum_{j=1}^s i_{j,4} y_j \\ \sum_{j=1}^s i_{j,5} y_j \\ \sum_{j=1}^s i_{j,6} y_j \end{bmatrix} \quad (5)$$

The dots represent where the rows and columns used to be. Notice that there is no longer any sign of the i_2 term. Now the input matrix is a 5x5 and the b and output matrices are reduced by a single row. From here the procedure is repeated (find the inverse of the 5x5 matrix and multiply both sides by the inverse matrix to get the remaining b coefficients). The coefficient for the removed term, b_1 in this case, is set to 0. Similarly, two terms can be removed from the equations by removing two sets of rows and columns, three terms are removed by removing three sets of rows and columns, etc. This continues until every possible combination of terms has been tested. The result will be a total of 77 sets of b coefficients. In order to determine which will give the best, the MSE is calculated the following:

$$MSE = \frac{1}{s} \sum_{i=1}^s (y_{d,i} - y_{c,i})^2 \quad (6)$$

The term $y_{d,i}$ represents the desired output for the sample set i. The other term, $y_{c,i}$, represents the calculated output for sample i using the inputs supplied for that sample and the coefficients calculated for the equation combination. The difference between these outputs is the prediction error of the neuron. By squaring the result, the error will always be positive and can easily be summed with the errors generated by the other sample input combinations. As each of the 77 neuron equations are created, the corresponding MSE is found. Rather than storing all 77 equations, the MSE's are calculated and compared as each potential neuron is created. Therefore, only the best combination is kept. After all 77 equations are tested; the algorithm creates the new neuron with the combination that resulted in the lowest MSE. This process is repeated until all possible combinations of two inputs to the layer have been tested.

Layers

The first layer of the network consists of a single node per input to the system. The input is handed to the first hidden stage without any alterations. After the input layer, the algorithm begins creating the hidden layers. These layers consist of neurons as shown in Fig. 1. Each neuron is a combination of two inputs to the layer. There are potentially a very large number of combinations of two inputs possible. We apply (4) once again, but with E set to 2. For example, with 5 inputs to a system there is a potential for 10 possible neurons. Realize that these 10 neurons then supply the inputs to the next layer. Therefore, the next hidden layer could have up to 45 neurons. If left unchecked, the network could grow very large very quickly. Therefore, the next step is to choose which of the neurons will be kept in the layer. To once again limit space, the algorithm has a numerical limit of two more neurons than the number of inputs to the layer. Therefore, the first $N = \text{inputs} + 2$ neurons created will be temporarily placed in the layer. If more

neurons are created, they are compared to the already saved neurons. Only the N neurons with the best MSE are kept.

Once the initial N neurons are chosen, the algorithm further limits the number of neurons in the layer by taking the average of the MSE of the surviving neurons. Any neuron with an MSE above this average is discarded. The idea is that forcing the system to keep N neurons could potentially mean that neurons with very poor MSE's could potentially become part of the final network. Also, if several neurons are giving very low MSE's, they may not all be necessary. More neurons can be kept by softening this rule. For example, it can be changed to any neuron with an MSE 10% higher than the average MSE will be discarded. An added benefit of this final test is that the number of neurons in a layer will typically be less than the number of neurons in the previous layer, limiting the number of layers in the final network and therefore keeping the network from growing to an unreasonable size.

The Final Network

Once the final set of neurons has been chosen, the algorithm must decide whether or not the layer is worth keeping. The algorithm allows the user/designer to set a maximum number of layers. This will be the absolute maximum. As stated in the previous subsection, the algorithm itself is designed to make each layer smaller than the layer before it. Once a new layer is created, the average MSE of its neurons is compared to the MSE of the previous layer. If there is no improvement, the layer is discarded and the algorithm creates a final layer. If average MSE is better, the layer is added to the network. If the new layer has only one neuron, the algorithm creates the final layer. Otherwise, the algorithm proceeds to create another layer with the output of the latest added layer as the input to the new layer. The output layer of the network consists of a single neuron that outputs the average of the outputs of the neurons from the last hidden layer.

Low Power Hardware Implementation

It can be concluded from eq. (1) and Fig. 2 that the hardware implementation of such an algorithm will mainly consist of a multiplier circuit for the product term along with a circuit for addition for the summation term. These two circuits in turn are derived from the basic logic gates such as AND, OR and NOT gate. In this paper we present a low power design of these basic gates along with a circuit for a multiplier and an adder. This allows us to exploit the property of the parallelism by replicating the basic operation. Proposed design uses power efficient basic hardware operations in the implementation of GMDH algorithm.

Hardware Implementation

The hardware architecture for any neural network algorithm can be implemented by using multiplication and addition circuits. Therefore, if we can optimize the design of the multiplier and the adder circuits then we can optimize the overall architecture.

This proposed hardware implementation GMDH Algorithm has been implemented by using complementary pass transistors. In this section we will analyze the proposed design by looking into aspects like power consumption, cost effectiveness, speed and wiring complexity. These results are obtained from Silos, LASI and Winspice simulations of the proposed designs. The interesting fact is the significant reduction in the number of nMOS and pMOS transistors employed in the design. This results in increasing the speed and a significant reduction in the power consumption and layout complexity. The reduced number of transistors directly corresponds to less wiring complexity.

Figure 3 (a) and (b) shows the traditional and proposed design of a multiplier circuit respectively. It can also be deduced from the layout of the proposed multiplication circuit that, the number of transistors employed for its implementation is drastically reduced as compared to its standard design. Figure 3 (a) shows two AND gates and one EXOR gate for implementing a simple signed multiplication function. Each AND gate is implemented by 3 nMOS and 3 pMOS transistors and the EXOR gate is implemented with 4

nMOS and 4 pMOS transistors. However, the proposed design implements a signed AND-function with four transistors (4 nMOS and 4 pMOS). Thus, the proposed design uses twelve transistors less for every signed multiplication circuit.

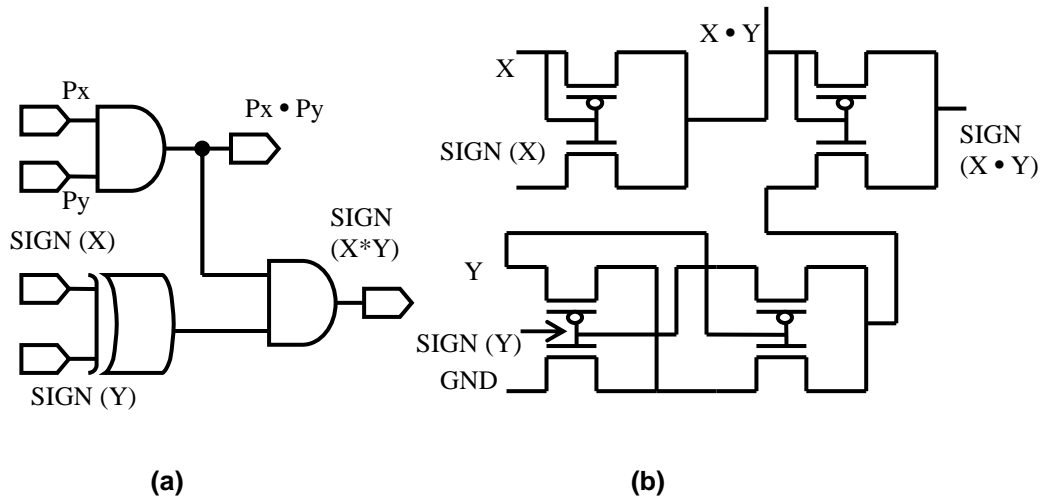
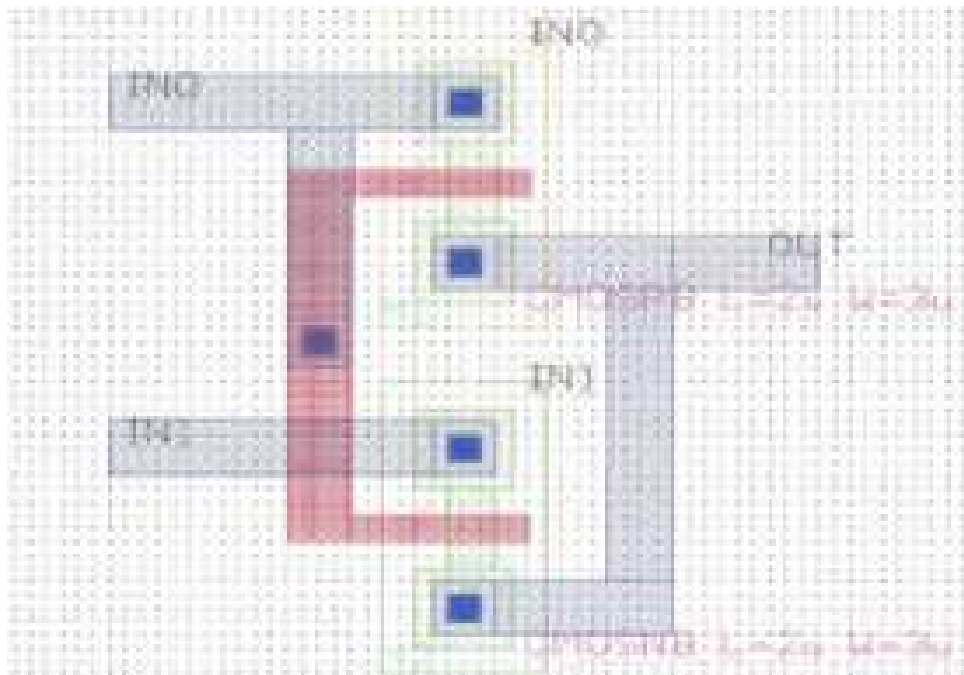
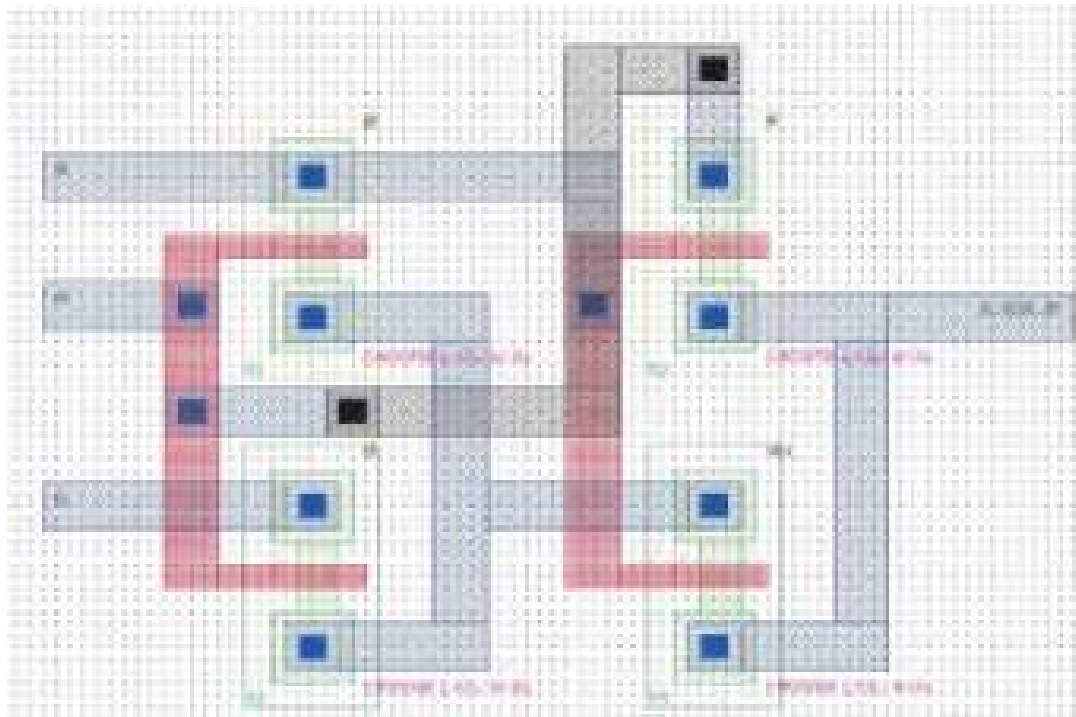


Figure 3 (a) and (b). Block Diagram for the Standard Design and the Proposed Low Power Design for the Signed Multiplication Circuit

The proposed design has been implemented in Fig. 4. Fig 5 shows the timing simulation for this implementation. This significant reduction in the transistor count increases the speed of the circuit and reduces the switching frequency which is an important parameter contributing towards the reduction in the dynamic power consumption. Another aspect to note in the proposed design is the absence of the explicit V_{DD} supply requirement in the proposed design of the multiplier circuit. Winspice simulation results for the layout analysis of proposed design of multiplication circuit showed excellent performance.



(a)



(b)

Figure 4 (a) and (b). Layout of the Low Power Design for the AND Gate and XOR Gate respectively

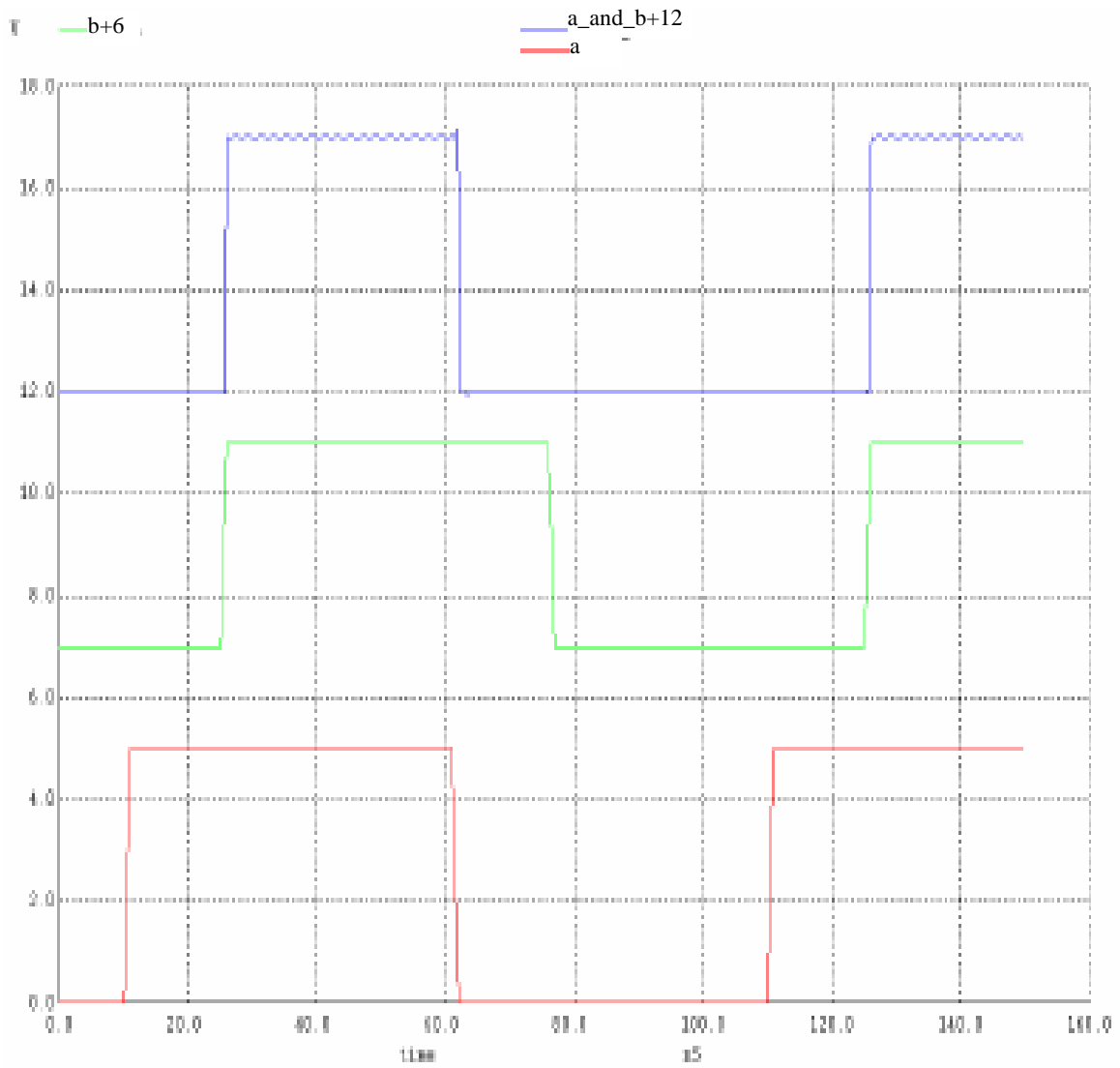


Figure 5 (a). Timing Analysis for the Low Power Design for the AND Gate (Y axis Represents the Voltage Level from 0.0 to 18.0 V at the interval of 2.0 volts and the x-axis represent the clock time)

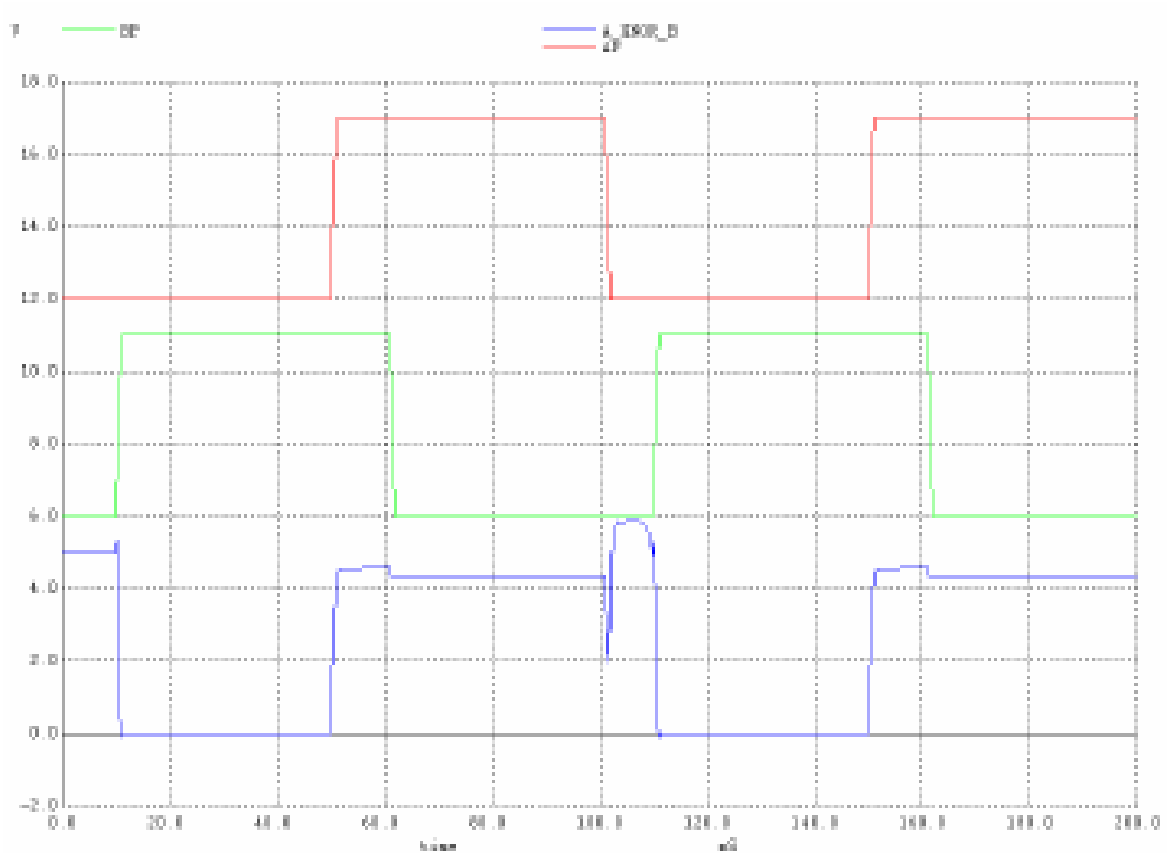


Figure 5 (a). Timing Analysis for the Low Power Design for the XOR Gate (Y axis Represents the Voltage Level from 0.0 to 18.0 V at the interval of 2.0 volts and the x-axis represent the clock time)

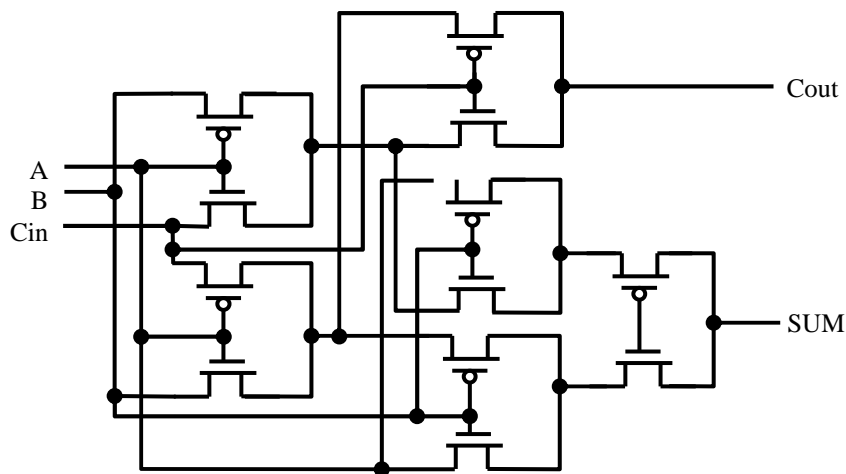


Figure 6. Block Diagram for the Low Power Design of the Full Adder

A standard CMOS design of the Full Adder circuit employs 28 transistors. A low power design for the Full Adder (See Fig. 6) was proposed by Al-Sheraidah [11]. This design uses only twelve gates against twenty-eight as in the standard design, shown in Fig. 7. This design is based on the transfer gate, which uses low power due to the absence of power supply. There are some trade-offs in this design in terms of

the voltage swing and the power consumption. Due to the absence of explicit V_{DD} and GND this circuit consistently consumes less power than the standard design of the adder circuit.

Power Consumption Analysis

The main reason for implementing the majority of contemporary high-complexity designs in static CMOS is the almost complete absence of power consumption in steady-state mode. According to (7) [12] [13], the average power dissipation P_v in digital CMOS circuits includes three distinct components.

$$P_v = P_{\text{short circuit}} + P_{\text{switching}} + P_{\text{leakage}} + P_{\text{static}} \quad (7)$$

The other three components of the (7) are denoted by (8), (9) and (10). It can be seen from the above equation that the dynamic power dissipation for digital CMOS circuits depends upon clock frequency, transition activity, node capacitance, short circuit current and the power supply V_{DD} .

$$P_{\text{switching}} = P_{\text{cap}} = \alpha \cdot C_L \cdot V_{DD}^2 \cdot f_{\text{clk}} \quad (8)$$

Here f_{clk} is the clock frequency, V_{DD} is the supply voltage, C_L is the load capacitance and alpha is the transition activity. In the proposed model as the number of transistors are reduced by 60% (instead of 20 transistor per signed multiplier we need 8 transistors per signed multiplier) the overall transition activity of transistors is also reduced by 60% accounting in the reduced power consumed. This capacitive load is originated from the capacitance between the gate and diffusion and the interconnecting metal and polysilicon layers in our drawn layouts. This can be substantially reduced by employing fewer transistors in our design and reducing the size of the transistor i.e. the length and the width of the channel to the minimum possible size.

$$P_{\text{leak}} = I_{\text{leak}} \cdot V_{DD} \quad (9)$$

Ideally, digital CMOS circuits should not exhibit any static power consumption. However, due to the non ideal sub threshold behavior of MOSFETs, there is a leakage current I_{leak} flowing from the positive power supply to the ground even in the static case resulting in the leakage power P_{leak} , which is given by the (9).

$$P_{\text{short}} = (1/2) \cdot \alpha \cdot (t_r \cdot I_{\text{short,max,r}} + t_f \cdot I_{\text{short,max,f}}) \cdot V_{dd} \cdot f_{\text{clk}} \quad (10)$$

where, $I_{\text{short,max,r}}$ is the peak of the current flowing from positive power supply to ground when n- and p-channel transistors are conducting simultaneously for a infinitely small moment during node transition and t_r/f is the fall and rise time of the node voltage. This short circuit power decreases with the decreasing switching activity 'alpha' and decreasing clock frequency 'fclk'.

It can be observed that in all the cases the power consumed directly depends upon V_{DD} i.e. the supply voltage. Due to the absence of the explicit V_{DD} and GND supply theoretically the dynamic power consumed will be negligible, whereas static power consumption will be responsible for the total power consumed by the device.

Applications

Whereas many algorithms today are designed with specific applications in mind, we have created an algorithm that we feel is flexible enough to fit into many different applications. In this section we have listed a few of the applications on which we have tested our algorithm. We will also discuss medical applications using sigmoid and radial basis function [14] GMDH type algorithms [18].

XOR paradigm

The most commonly used experiment for testing a neural network is the XOR function [15], so we will start with that particular challenge. The simulation found that a single hidden node with the following equation was enough:

$$y=0+0i_{n1}+0i_{n2}+1i_{n1}^2+1i_{n2}^2-2(i_{n1})(i_{n2}) \quad (11)$$

Other networking learning algorithms such as back propagation take thousands of iterations and still have quite large errors [15]. Wu et al tested two methods, BPNET and BPNET-CS, with the XOR problem. BPNET is a basic back propagation algorithm that includes an additional term in calculating the weights that takes into account previous improvements in the output. BPNET-CS is also a back propagation algorithm, but it uses a more complex error based calculation to speed up the training process. The BPNET still has a large error with over 10000 iterations. The BPNET-CS method has about 2.4% error after 700 iterations. With our learning algorithm, the equation was quickly found after calculating and testing the 77 possible combinations of the six neuron terms. No error is present after the final equation for the neuron is selected. Fig. 7 shows the architecture based on GMDH as compared to other previously proposed architectures for the XOR problem.

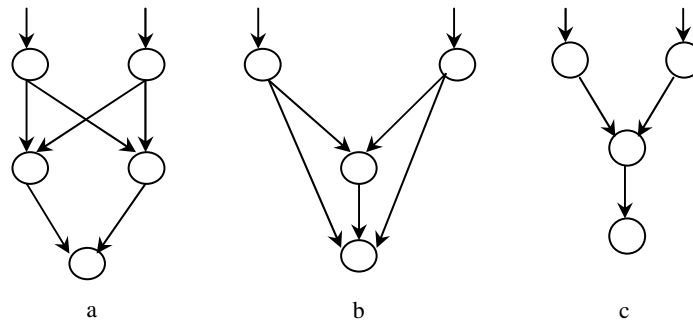


Figure 7. Various Neural Network Architectures for Implementing an XOR Function
(a) Based on multilayer perceptron **(b)** based on feedforward architecture **(c)** based on GMDH

Product Usage

This experiment is based on a problem in the Draper text [10]. The problem uses relative urbanization, educational level, and relative income as the independent input variables, and the usage of a certain product as an output. Draper attempted to predict the output using Regression analysis. The network produced by our algorithm had a total of 4 layers and 7 neurons. The output had approximately 1.47% error with the training data. The error for the equation developed by Draper was approximately 1.91% error.

Predict Aggregate Quality Parameter SPR From The Four Identified Factors, DEN, Ir, Qz, and spr

The data for this application are originally from Hogstrom [15] and quoted in Huang. Huang [16] evaluated them with a single layer back propagation artificial neural network. For this data, the input values represent density (DEN), point load (Ir), quartz content (Qz), and brittle mineral content (spr). These four factors determine the quality parameters known as the impact value (SPR). Along with the four inputs, 56 training set samples were supplied. Since Huang's [16] purpose was to prove that a single hidden layer in a back propagation network was enough to supply acceptable results, the same restriction was placed on the network software. The network was limited to a single stage. The same testing method was used for this analysis as Huang used. One input combination was taken out of the 56. The other 55 values would be used to train the network, and then the input combination taken out would be used for testing. The error for each combination corresponds to using that line as the test combination. Our algorithm produced only a 5.62% error while the back propagation method resulted in 5.75%.

Medical Image Recognition

Automatic identification of organs in the medical images and their volume calculation are important issues in Computer Aided Diagnosis (CAD). It is possible to extend the architecture discussed in section 4 to

more complex types of neurons such as sigmoid or radial basis function type neurons. GMDH-type neural networks with sigmoid functions were applied for the identification of the liver in CAT (Computer Aided Tomography) scan images and it is shown that this algorithm is very useful for medical image recognition. The GMDH-type neural networks with sigmoid functions have the abilities of self-selecting useful input variables and of self-organizing optimum neural network architecture [8]. The structural parameters such as the number of the layers, the number of the neurons in the hidden layers, the useful input variables etc. are automatically determined so as to minimize the error criterion defined as AIC (Akaike's information criterion) [14]. Fig. 8 shows the CAT scan of the liver and Fig. 9 shows the extracted liver region using the GMDH algorithm.

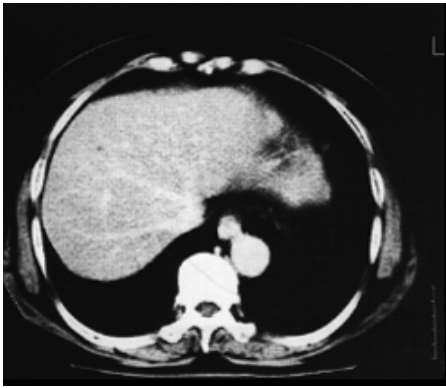


Figure 8. CAT scan image of the Liver

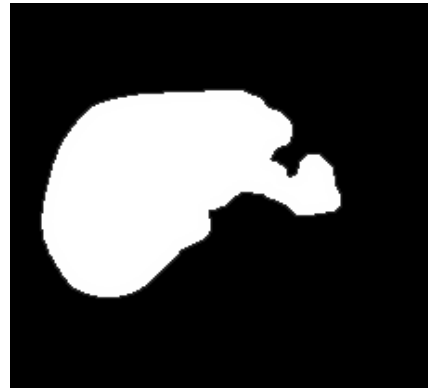


Figure 9. Liver region extracted using the GMDH algorithm

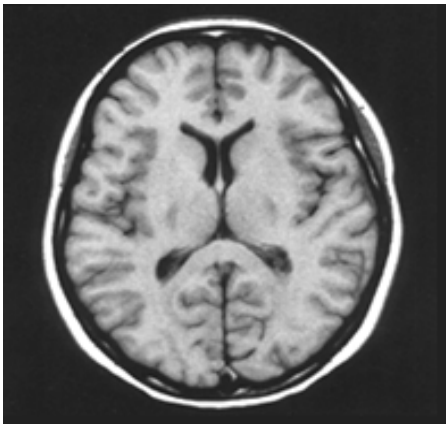


Figure 10. MRI scan image of the Brain



Figure 11. Brain region extracted using the GMDH algorithm

The GMDH-type neural networks with radial basis functions were applied to medical image recognition of the brain. Fig. 10 shows the MRI scan of the brain while Fig. 11 shows the result of applying GMDH for extracting the brain region.

Conclusions

We have proposed and developed a GMDH type training algorithm that produces the smallest, most accurate neural network possible. Since our method does not require much iteration to calculate the

weights of each node, it can produce a network very quickly. We do not contend that our algorithm will give the best results in all cases, but our results show that it is more flexible, accurate, and faster than many algorithms currently available. As shown by various medical applications the algorithm can be extended to other type of neurons such as the sigmoid or radial basis function type neurons.

References

- [1] J. Seul, K. Sung, Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems, *IEEE Transactions on Industrial Electronics*, 54(1), 2007, 265-271.
- [2] C. Chao-Ming, L. Chih-Min, C. Ching-Tsan, D. S. Yeung, Hardware implementation of CMAC neural network using FPGA approach, *IEEE International Conference on Machine Learning and Cybernetics*, 4, 2007, 2005-2011
- [3] A. S. Pandya and R. B. Macy, *Pattern Recognition using Neural Networks in C++*, (IEEE Press and CRC Press, 1995).
- [4] A.G. Ivakhnenko, Heuristic self-organization in problems of engineering cybernetics, *Automatica*, 6(2), 1970, 207-219.
- [5] M.C. Acock, Y.A. Pachepsky, estimating missing weather data for agricultural Simulations using group method of data handling, *Journal of Applied Meteorology*, 39(2), 2000, 1176-1184.
- [6] T. Kondo, A.S. Pandya, J.M. Zurada, Logistic GMDH-type neural networks and their application to the identification of the X-ray film characteristic curve, *Proc. of IEEE International Conference on System, Man, and Cybernetics*, 1999, 437-442.
- [7] T. Kondo, A. S. Pandya and H. Nagashino , GMDH-type neural network algorithm with a feedback loop for structural identification of RBF neural network, *International Journal of Knowledge-Based Intelligent Engineering Systems*, 11(7), 2007, 157-168.
- [8] T. Kondo, J. Ueno and A.S. Pandya, Multi-layered GMDH-type neural networks with radial basis functions and their application to the 3-dimensional medical image recognition of the liver, *Journal of Advanced Computational Intelligence*, 11(7), 2007, 157-168
- [9] T. Kondo, A.S. Pandya, GMDH-type neural network algorithm with sigmoid functions, *International Journal of Knowledge-Based Engineering Systems*, 7(4), 2003, 198-205.
- [10] N.R. Draper, H. Smith, *Applied Regression Analysis*, (New York: John Wiley & Sons, Inc., 1966).
- [11] A. Agarwal, A Low Power Design of an ALU, *MS Thesis, Florida Atlantic University* 2003.
- [12] A. Agarwal, A. S. Pandya, Y.U. Lho, Low power high frequency data transfer for a RISC and CISC architecture, *The International Journal of the Korean Institute of Maritime Information and Communication Science*, 10(2), 2006, 321-327
- [13] A. Agarwal, A. S. Pandya, Y. U. Lho, A novel low power design of an ALU using ad-hoc techniques, *International Journal of Fuzzy Logic and Intelligent Systems*, 5(2), 2005, 102-117
- [14] H. Akaike, A new look at the statistical model identification, *IEEE Trans. Automatic Control*, 1(6), 1974, 716-723.
- [15] P. Wu, S. Fang, H. Nuttle, Efficient neural network learning using second order Information with Fuzzy Control, *Neurocomputing*, 43(1), 2002, 197-217.

- [16] Hogstrom, K Study on Strength Parameters for aggregates from south western Swedish rocks, Research Report, Chalmers University of Technology, Gotzborg, Sweden, 1994.
- [17] Huang Y, Application of Artificial neural networks to predictions of aggregate quality parameters, *International Journal of Rock Mechanics and Mining Sciences*, 36(4), 1999, 551-561.
- [18] T. Kondo, Revised GMDH type neural networks with radial basis functions and their application to medical image recognition of stomach, *System Analysis Modelling Simulation*, 43(10), 2003, 1363-1376.