A Green Granular Neural Network with Efficient Software-FPGA Co-designed Learning

Huaiyuan Chu, Yanqing Zhang
Department of Computer Science
Georgia State University
Atlanta, GA 30302-5060, USA
chuhy4@gmail.com, yzhang@gsu.edu

***Abstract*—A novel green granular neural network (GGNN) with new fast software-FPGA co-designed learning is developed to reduce both CO_2 emissions and energy consumption more effectively than popular neural networks with the traditional software-CPU-GPU-based learning. Different from traditional tedious CPU-GPU-based training algorithms using gradient descent methods and other methods such as genetic algorithms , the software-FPGA co-designed training algorithm may quickly solve a system of linear equations to directly calculate optimal values of hyperparameters of the GGNN. Initial simulation results indicates that the FPGA equation solver code ran faster than the Python equation solver code. Therefore, implementing the GGNN with software-FPGA co-designed learning is feasible. In addition, the shallow high-speed GGNN is explainable because it can generate interpretable granular If-Then rules. In the future, The GGNN will be evaluated by comparing with other machine learning models with traditional software-based learning in terms of speeds, model sizes, accuracy, CO_2 emissions and energy consumption by using popular datasets. New algorithms will be created to divide the inputs to different input groups that will be used to build different small-size GGNNs to solve the curse of dimensionality. Additionally, the explainable green granular convolutional neural network will be developed by using the GGNNs as basic building blocks to efficiently solve image recognition problems.**

***Index Terms*—granular neural networks, FPGA, software-hardware co-designed learning, green computing**

I. Introduction

In recent years, deep neural networks have been effectively used in computer vision applications. However, a major problem is that traditional tedious CPU-GPU-based training algorithms using gradient descent methods and other methods such as genetic algorithms take huge amount of training time, generate much CO_2 emissions and waste a lot of energy.

In recent years, new green machine learning (ML) systems have been made to reduce both CO_2 emissions and computational energy consumption. For instance, the AutoML framework for different methods such as neural architecture search (NAS), and automated pruning and quantization is used to build efficient on-device ML systems with low energy consumption and low CO_2 emissions by measuring GPU hours and the estimated CO_2 emission amount CO_2e [5]. Since CO_2e is proportional to the total computational power p_t: $CO_2e = 0.954p_t$ [6], effectively reducing training times results in greatly reducing both energy consumption and CO_2 emissions.

This material is based upon work supported by the National Science Foundation under Grant No. 2234227.

Currently, popular ML systems running on CPUs and GPUs generate a lot of CO_2 emissions and also waste much energy because (1) tedious traditional training algorithms such as gradient descent algorithms and genetic algorithms take huge amount of time to optimize billions of hyperparameters, and (2) CPUs and GPUs are not green effective and not energy efficient. In summary, an urgent challenge is developing a novel ML system with high-speed non-traditional training algorithms running on the green and energy efficient hardware to significantly reduce both CO_2 emissions and energy consumption.

We will develop the novel green granular neural network (GGNN) with new fast FPGA-based training algorithm to effectively reduce both CO_2 emissions and energy consumption more effectively than the CPU-GPU-based training algorithms. We will also develop a novel high-speed FPGA-based incremental transfer learning algorithms to greatly reduce both CO_2 emissions and energy consumption for real-time green computing applications.

Based on the successful implementation of the FPGA-based direct linear equation solver [7], [8], [14], the high-speed FPGA-based direct linear equation solver can be used to quickly generate optimal hyperparameters in just one epoch for the new GGNN in a real-time manner. For example, the Questa*-Intel FPGA Edition Software provides the FPGA design simulation that involves generating simulation files, compiling simulation models, running the simulation, and viewing the results. We will use FPGA software simulation systems to implement the high-speed FPGA-based direct linear equation solver. The goal is to develop more effective and faster hardware-based hyperparameter optimization algorithms with a fast direct linear equation solver for training a new GGNN.

II. The Highly Efficient FPGA-based GGNN

Two major ways of effectively reducing CO_2 emissions include (1) develop novel high-speed non-traditional training algorithm to significantly reduce training times, and (2) use novel computational hardware with low CO_2 emissions.

Approach 1: Firstly, we will develop fast training algorithms different from traditional tedious hyperparameter optimization methods such as gradient decient methods and genetic algorithms. To minimize $Q = 1/2\sum_{k=1}^{N}(y_k - Y_k)^2$ where y_k is an output of a ML model, and Y_k is a correct output for $k = 1, 2, \ldots, N$, we have $\frac{\partial Q}{\partial \theta_i} = 0$ for $i = 1, 2, \ldots, n$ where θ_i is a parameter. If we cannot directly calculate θ_i, then we have to use the tedious training algorithms such as gradient descent algorithms and genetic algorithms. Differently, we will develop the novel green granular neural network (GGNN) with the fast training algorithm that can quickly calculate parameters θ_i for $\frac{\partial Q}{\partial \theta_i} = 0$ without slow training epoch by epoch. Importantly, current $\theta_i(t)$ can be updated efficiently and incrementally to generate $\theta_i(t+1)$ by only using new training data (in other words, each training data is used only once to calculate the parameters for really fast real-time training). Such high-speed incremental transfer learning methods are useful for real time ML applications such as smart apps on mobile devices and federated edge learning systems with low pollution, low energy consumption and high speed.

Approach 2: Secondly, we will develop the novel GGNN with new fast FPGA-based training algorithm to reduce CO_2 emissions more effectively than the CPU-GPU-based training algorithms. Popular CPUs and GPUs generate much more CO_2 emissions and run less efficiently than the field programmable gate array (FPGA) [4, 5]. For instance, the new FPGA-based massive parallel data processing system can reduce CO_2 emissions by around 50% [5]. FPGA is a lightweight hardware with low CO_2 emissions and low energy consumption [6] for quickly solving a system of linear equations. For example, on a Xilinx Vertex 6 FPGA (200MHz), the minimum latency of the FPGA-based direct linear equation solver was lower than 5 microseconds for a linear system of equations of order 32 [7]. Thus, it is feasible to use FPGA to implement the new FPGA-based training algorithm.

III. A Fast GGNN with Software-FPGA Co-Designed Learning

A. Granular Sets

Different sets dealing with uncertainty of data and information, such as the fuzzy set [22], the neutrosophic fuzzy set [25], the intuitionistic fuzzy set [23], and Pythagorean fuzzy set [24], were defined. A new granular set is defined as follows to be used to build a new granular neural network.

Definition 1. Let X be a nonempty set. A granular set A in X is defined as $A = \{\langle x, \mu_A(x), \nu_A(x), \phi_A(x), \varphi_A(x), \theta_A(x), \vartheta_A(x)\rangle : x \in X\}$, where (1) $\mu_A(x)$ is degree of membership of x for $0 \le \mu_A(x) \le 1$, (2) $\nu_A(x)$ is degree of non-membership of x for $0 \le \nu_A(x) \le 1$, (3) $\phi_A(x)$ is certain degree of $\mu_A(x)$ for $0 \le \phi_A(x) \le 1$, (4) $\varphi_A(x)$ is uncertain degree of $\mu_A(x)$ for $0 \le \varphi_A(x) \le 1$, (5) $\theta_A(x)$ is certain degree of $\nu_A(x)$ for $0 \le \theta_A(x) \le 1$, and (6) $\vartheta_A(x)$ is uncertain degree of $\nu_A(x)$ for $0 \le \vartheta_A(x) \le 1$, where $0 \le \mu_A(x) + \nu_A(x) \le 1$, $0 \le \phi_A(x) + \varphi_A(x) \le 1$, and $0 \le \theta_A(x) + \vartheta_A(x) \le 1$.

Meaningful linguistic values, such as $very\ slow$, $about\ 25$, $around\ 200$, can be represented by the granular sets that are used to build interpretable granular fuzzy If-Then rules. For example, an explainable granular If-Then rule is If x_1 is low and x_2 is $around\ -1000$, Then an output is $high$.

B. Software-FPGA Co-designed Learning

A new GGNN with software-FPGA co-designed learning using training data is developed as shown in Fig. 1. It uses the software-based learning system to compute the coefficients for a linear system of hyperparameter equations, then uses the fast FPGA-based learning system to optimize the hyperparameters, and finally builds a GGNN model for prediction.

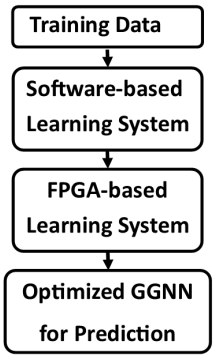

Fig. 1. A GGNN with Software-FPGA Co-Designed Learning

For convenience, an N-record relational database has n numerical input fields x_i for $i = 1, 2, ..., n$, and one numerical output field y. Now the problem is how to build a GGNN using given N records in the relational database. Here, granular sets are used as basic granules in granular partitions of the input variables x_i for $i = 1, 2, ..., n$ and the output variable y. The interval $[a_i, b_i]$ of x_i are partitioned into $m_i - 1$ intervals ($a_{i1} \le x_i \le a_{i2}$, $a_{i2} \le x_i \le a_{i3}$, ..., $a_{i(m_i-1)} \le x_i \le a_{im_i}$). So m_i granules A_{ij} are used to cover the $m_i - 1$ intervals for $i = 1, 2, ..., n$, $j = 1, 2, ..., m_i$. The granules A_{ij} are defined by granular sets such as a fuzzy set [22]. After the above granulation of x_i for $i = 1, 2, ..., n$, there are G data base granules for $G = \prod_{i=1}^{n}(m_i - 1)$. For each data base granule, a GGNN with an output $g(x_1, x_1, ..., x_n)$ is constructed by using two input granular sets covering one interval of x_i and 2^n output granular sets of y. So y has 2^n granular sets B_k for $k = 1, 2, ..., 2^n$.

The granular rule base has 2^n granular IF-THEN rules for one database granule such that $IF\ x_1\ is\ A_{1j_1}\ and\ ...\ x_n\ is\ A_{nj_n}\ THEN\ y\ is\ B_k$ for $j_i \in 1, 2$, $i = 1, 2, ..., n$, and $k = 1, 2, ..., 2^n$.

A database granule has $K = \prod_{i=1}^{n} k_i$ records totally if an input x_i has k_i values for $i = 1, 2, ..., n$ in the database granule, and an output y has K corresponding values y_k for $k = 1, 2, ..., K$. The optimization function for the database granule is to minimize $Q = \frac{1}{2}\sum_{k=1}^{K}[y_k - g(x_{1_k}, x_{2_k}, ..., x_{n_k})]^2$. Based

on $\frac{\partial Q}{\partial p_j} = 0$ for the GGNN, we can get a linear system of M-hyperparameter equations for $k = 1, 2, ..., M$ for $M = 2^{n+1}$:

$$T_1^k q_1 + T_1^k q_2 + ... + T_M^k q_M = \psi_k \quad (1)$$

Now we can solve the linear system of M-hyperparameter equations to directly get optimal M hyperparameters q_k of the GGNN for $k = 1, 2, ..., M$.

Based on the successful design of the FPGA-based linear equation solver [7], [8], [14], it is feasible to use the same architecture of the FPGA-based linear equation solver to solve equation (1) to get optimized hyperparameters q_k for $k = 1, 2, ..., M$.

The major merits of the granular constructive learning method are (1) quickly optimize parameters using predefined formulas, and (2) discover meaningful granular rules from training data.

The direct hyperparameter optimization algorithm for a highly efficient GGNN with the efficient granular knowledge transfer learning and incremental learning is given below.

FPGA-based Training Algorithm with Direct Hyperparameter Optimization
Begin
Input: N training data.
Output: A GGNN with discovered granular knowledge with optimized hyperparameters.

Step 1 (Using Software to Pre-calculate Coefficients for a Linear System of Hyperparameter Equations): use software to calculate coefficients such as T_1^k, T_2^k, ..., T_M^k of a linear system of hyperparameter equations based on given N training data.

Step 2 (Using FPGA to Solve the Linear System of Hyperparameter Equations): Input the coefficients to a pre-designed FPGA, then use FPGA-based linear equation solver to calculate optimal hyperparameters q_k for $k = 1, 2, ..., M$ of the linear system of hyperparameter equations. The optimized hyperparameters are used to build a granular knowledge base with meaningful granular If-Then rules.

Step 3 (Using a FPGA-based GGNN with Newly Discovered Granular Knowledge for an Application): Use the newly discovered granular knowledge to build a FPGA-based GGNN, then use the FPGA-based GGNN for an application.
End

The FPGA-based direct hyperparameter optimization algorithm for a GGNN is highly efficient because of its four high-speed hybrid software-hardware-based steps. Step 1 uses software to calculate coefficients only one time based on all available training data. Step 2 uses fast hardware-based linear equation solver to quickly get optimal hyperparameters that are used to make granular knowledge. Step 3 uses the fast FPGA-based GGNN for a real-time ML application.

IV. Simulations for Software-based Learning Methods

To compare an artificial neural network (ANN) and the GGNN using a fuzzy set (a special granular set), simulations using two different functions are done. The first 3-input-1-output benchmark function f_k^1 [18]–[21] is given below:

$$f_k^1 = (1 + x_k^{0.5} + y_k^{-1} + z_k^{-1.5})^2. \quad (2)$$

The training data set with $8,000$ training data is generated by f_k^1 shown in equation (2) such that $x_k^{tr} = 1.0 + \lfloor \frac{k}{400} \rfloor$, $y_k^{tr} = 1.0 + \lfloor \frac{k}{20} \rfloor$, $z_k^{tr} = 1.0 + k \ mod 20$, where the operator mod is used, $f_k^1 \in [4.248, 55.833]$, and $k = 0, 1, ..., 7,999$. A testing data set with $6,859$ testing data is generated by f_k^1 such that $x_j^{te} = 1.5 + \lfloor \frac{j}{361} \rfloor$, $y_j^{te} = 1.5 + \lfloor \frac{j}{19} \rfloor$, $z_j^{te} = 1.5 + j \ mod 19$, where the operator mod is used, $j = 0, 1, ..., 6,858$. $8,000$ training data are distributed in 27 subspaces, but data in 16 subspaces are used to train both ANNs and $GGNN$ (i.e., no training data in 11 other subspaces like missing data in the subspaces). $6,858$ testing data are distributed in all the 27 subspaces to compare ANNs and $GGNN$.

Tables 1 to 3 show that $GGNN$ outperforms both 10-Layer ANN and 20-Layer ANN in terms of the prediction Mean Square Error (MSE), and the prediction Root Mean Square Error (RMSE) for 100, 500, and $1,000$ training epochs.

TABLE I
Function Prediction Performance Comparison between ANNs and the GGNN for f^1 for 100 training epochs.

Neural Network	MSE	RMSE
10-Layer ANN	58.22	7.63
20-Layer ANN	63.44	6.88
$GGNN$	47.31	6.88

TABLE II
Function Prediction Performance Comparison between ANNs and the GGNN for f^1 for 500 training epochs.

Neural Network	MSE	RMSE
10-Layer ANN	55.68	7.46
20-Layer ANN	76.99	8.77
$GGNN$	46.38	6.81

TABLE III
Function Prediction Performance Comparison between ANNs and the GGNN for f^1 for $1,000$ training epochs.

Neural Network	MSE	RMSE
10-Layer ANN	51.16	7.15
20-Layer ANN	53.66	7.33
$GGNN$	46.71	6.83

V. Simulations for FPGA-based Learning Methods

Once we calculated coefficients as T_1^k, T_2^k, ..., T_M^k, we can solve equation (1) by simply using matrix inversion method. However, matrix inversion is, by its nature, not hardware-friendly. Many algorithms rely on division which requires huge resources on FPGA. Also, we usually need to fix the matrix size in prior to feeding numbers to the hardware. The first problem has been a hot topic in the FPGA community, and the

second problem can be solved by HLS (High-Level Synthesis) [9].

Based on previous sections, if we have n input parameters, T_1^k, ..., T_M^k will form a square matrix with $2^{n+1} \times 2^{n+1}$. There have been some researches focusing on FPGA-based matrix inversion for the past decades [14]–[16], such as steepest descent method [11], QR decomposition [12] and Gauss Jordan method [13], etc. The current method we use is LUP (LU factorization with partial pivoting). Fig. 2 shows an example for a 4×4 matrix. Following the color order, we can easily decompose a given matrix in $A = LU$. And the inverse of an upper or lower triangular matrix is easy to compute, since U^{-1} is also an upper triangular matrix [17].

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{44} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

$$= \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21}u_{11} & l_{21}u_{12}+u_{22} & l_{21}u_{13}+u_{23} & l_{21}u_{14}+u_{24} \\ l_{31}u_{11} & l_{31}u_{12}+l_{32}u_{22} & l_{31}u_{13}+l_{32}u_{23}+u_{33} & l_{31}u_{14}+l_{32}u_{24}+u_{34} \\ l_{41}u_{11} & l_{41}u_{12}+l_{42}u_{22} & l_{41}u_{13}+l_{42}u_{23}+l_{43}u_{33} & l_{41}u_{14}+l_{42}u_{24}+l_{43}u_{34}+u_{44} \end{bmatrix}$$

Fig. 2. LU factorization on a 4×4 matrix [10]

If we have 2 input parameters, we will form an 8×8 square matrix. We generate a random matrix A and an 8×1 vector b, then we want to solve the linear equation $Ax = b$. For example, We used the code of MATLAB and the FPGA code [8] to generate the same solution: (-0.5830, -0.7406, 0.8177, -0.0643, 0.5189, -0.0649, 0.5263, 0.4562) for the following equation:

1.4432	1.2248	-2.7408	4.0488	1.0284	-4.1448	-2.6272	1.7914	-4.0129
-1.2139	0.8704	-3.2929	4.7975	2.1122	-2.3752	-0.4115	-1.0448	-2.3813
3.1158	-2.9226	-2.7234	-0.6113	-2.7825	3.0101	4.6309	-1.3256	-1.6464
0.3283	-1.9875	-0.6430	-3.8888	-3.8258	-4.7078	0.4681	4.8798	1.7973
-1.4927	-0.2908	-1.8890	-2.4194	-2.0332	4.2885	0.2114	-4.6226	-3.6345
4.3900	-2.6951	4.2338	-0.9128	-1.8122	2.3033	-2.6841	3.8517	2.2123
3.7594	3.4431	-0.6979	0.9490	-0.7583	-0.1139	-0.1110	4.1329	-3.9324
0.5016	-3.0524	-3.1518	-2.3779	0.0786	0.7853	1.2406	2.9618	1.5376

Furthermore, we used a MATLAB code, a FPGA code and a Python code to test their running times on for the matrix inversion with different linear systems of hyperparameter equations of different orders (i.e., 8, 16, 32, 64, and 100). For each case, we create 20 complex square matrices of different orders. Table 1 shows running times that are measured in 10^{-4}s. All the tests are running on the same computer. The FPGA code ran faster than the Python code. The FPGA code ran faster than the MATLAB code except for a case of order of 16.

In addition, the FPGA code is effective to reduce both energy consumption and CO_2 emissions because the short execution time of the FPGA code results in a small computational power consumption p_t for $CO_2e = 0.954p_t$ [6] . Importantly, a FPGA hardware will be much faster than a software-based equation solver to reduce both energy consumption and CO_2 emissions more effectively.

TABLE IV
RUNNING TIMES OF THE THREE CODES

Method	8	16	32	64	100
MATLAB	0.689	1.232	3.989	17.298	62.410
Python	1.139	1.581	7.057	30.147	76.581
FPGA	0.638	1.313	2.803	6.207	12.284

Based on LUP, we can write the corresponding C program. To generate feasible Verilog scripts, we can use Vivado HLS to transform the C program to Verilog code and simulate it in the software. Therefore, the new software-FPGA-based learning method is feasible and useful for implementing the new fast FPGA-based GGNNs.

VI. CONCLUSIONS

Initial simulation results indicates that the FPGA equation solver code ran faster than the Python equation solver code. In additon, the GGNN may perform more accurately than a traditional neural network. Therefore, it is feasible to make a novel software-FPGA co-designed GGNN to reduce both CO_2 emissions and energy consumption more effectively than the CPU-GPU-based neural networks. Since FPGA is a high-speed light-weight hardware with low CO_2 emissions and low energy consumption, the FPGA is used to quickly solve a system of linear equations to directly calculate optimal values of hyperparameters of the shallow GGNN.

VII. FUTURE WORKS

In the future, the GGNN with the software-FPGA co-designed learning will be evaluated by comparing with other machine learning models with traditional software-CPU-GPU co-designed learning in terms of speeds, model sizes, accuracy, CO_2 emissions and energy consumption by using popular datasets. New intelligent algorithms will be developed to find out optimal or near optimal sub-spaces on which accurate GGNN models will be built.

A GGNN with a large number of inputs has the curse of dimensionality. New algorithms will be created to divide the inputs to different input groups that will be used to build different small-size GGNNs to solve the problem.

The explainable green granular convoluational neural network (GGCNN) will be devleoped by using the GGNNs as basic building blocks to efficiently solve image recognition problems. The new GGCNN consists of convolutional layers, activation layers, pooling layers, and the fast GGNN with the software-FPGA co-designed learning.

The shallow high-speed GGNN is explainable because it can generate interpretable granular If-Then rules. In the future, we will use different granular sets with different nonlinear membership functions, and then select the best one to improve performance (accuracy, AUC, F1-score, etc.) of the GGNN.

After this FPGA-based learning software is successful, a high-speed FPGA hardware based direct linear equation solver will be implemented for building an efficient GGNN and an efficient GGNN to significantly reduce both CO_2 emissions and energy consumption.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, . Kaiser, and I. Polosukhin, "Attention is all you need," 2017 NeurIPS, pp. 5998–6008.

[2] K. He and X. Zhang and S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 CVPR, pp. 770–778.

[3] A. Krizhevsky and I. Sutskever and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012 NeurIPS, pp. 1097–1105.

[4] Y. LeCun and Y. Bengio and G.E. Hinton, "Deep learning," Nature, vol. 521, pp. 436–444, 2015.

[5] Y. H. Cai and J. Lin and Y. Lin and Z. Liu and H. Tang and H. Wang and L. Zhu and S. Han, "Enable Deep Learning on Mobile Devices: Methods, Systems, and Applications," ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 27, Issue 3, Article 20, pp. 1–50, 2021.

[6] E. Strubell and A. Ganesh and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," the 57th Annual Meeting of the Association for Computational Linguistics (ACL), 2019.

[7] L. Miller, "Adaptive Beamforming for Radar: Floating-Point QRD+WBS in an FPGA," 2014.

[8] M. Ruan, "Scalable Floating-Point Matrix Inversion Design Using Vivado High-Level Synthesis," 2017.

[9] M. Ruan, https://www.xilinx.com/.

[10] https://scm_mos.gitlab.io/math/matrix-decomposition/.

[11] M. Ruan, "Fpga design and implementation of direct matrix inversion based on steepest descent method," 2007 50th Midwest Symposium on Circuits and Systems, pp. 1213–1216.

[12] A. Irturk, S. Mirzaei and R. Kastner, "An efficient FPGA implementation of scalable matrix inversion core using QR decomposition," 2009.

[13] S. Chetan, K.S. Sourabh, V. Lekshmi, S. Sudhakar, J. Manikandan, "Design and Evaluation of Floating point Matrix Operations for FPGA based system design," Procedia Computer Science, vol. 171, pp. 959–968, 2020.

[14] Z. Jiang and S. A. Raziei, "An efficient FPGA-based direct linear solver," 2017 IEEE National Aerospace and Electronics Conference (NAECON), pp. 159–166, 2017.

[15] M. Karkooti, J. R Cavallaro and C. Dick, "FPGA implementation of matrix inversion using QRD-RLS algorithm," Asilomar Conference on Signals, Systems, and Computers, 2005.

[16] A. Irturk, B. Benson, S. Mirzaei and R. Kastner, "An FPGA design space exploration tool for matrix inversion architectures," 2008 Symposium on Application Specific Processors, pp. 42–47, 2008.

[17] https://en.wikipedia.org/wiki/Triangular_matrix.

[18] T. Kondo, "Revised GMDH algorithm estimating degree of the complete polynomial," Trans. SOC. Instrument and Contr. Engineers, vol. 22, no. 9, pp. 928–934, 1986.

[19] M. Sugeno and G. T. Kang, "Structure identification of fuzzy model," Fuzzy Sets and Systems, vol. 28, no. 1, pp. 15–33, 1988.

[20] H. Takagi and I. Hayashi, "NN-driven fuzzy reasoning," The International Journal of Approximate Reasoning, vol. 5, no. 3, pp. 191–212, 1988.

[21] J.S.R. Jang, "ANFIS: adaptive-network-based fuzzy inference system," IEEE Transactions on Systems, Man, and Cybernetics, vol. 23, no. 3, pp. 665–685, 1991.

[22] L. A. Zadeh, "Fuzzy sets," Information and Control, vol. 8, no. 3, pp. 338–353, 1965.

[23] K. T. Atanassov, "Intuitionistic fuzzy sets," Fuzzy Sets and Systems, vol. 20, no. 1, pp. 87–96, 1986.

[24] R. Yager, "Pythagorean membership grades in multicriteria decision making," IEEE Transactions on Fuzzy Systems, vol. 22, no. 4, pp. 958–965, 2013.

[25] F. Smarandache, "Neutrosophy: Neutrosophic Probability, Set, and Logic: Analytic Synthesis & Synthetic Analysis," American Research Press, 1998.