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

Anonymous Author(s)
Affiliation
Address
email

Abstract

Different from traditional tedious CPU-GPU-based training algorithms using gradient descent methods, the software-FPGA co-designed learning algorithm is created to quickly solve a system of linear equations to directly calculate optimal values of hyperparameters of the green granular neural network (GGNN). To reduce both CO_2 emissions and energy consumption effectively, a novel green granular convolutional neural network (GGCNN) is developed by using a new classifier that uses GGNNs as building blocks with new fast software-FPGA co-designed learning. Initial simulation results indicates that the FPGA equation solver code ran faster than the Python equation solver code. Therefore, implementing the GGCNN with software-FPGA co-designed learning is feasible. In the future, The GGCNN will be evaluated by comparing with a convolutional neural network (CNN) with the traditional software-CPU-GPU-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.

1 Introduction

In recent years, deep neural networks such as a Convolutional Neural Network (CNN) have been effectively used in different applications. However, a major problem is that traditional tedious CPU-GPU-based training algorithms using gradient descent methods take huge amount of training time, generate much CO_2 emissions and waste a lot of energy. For instance, a popular CNN needs a large number of training epochs for very slow hyperparameter optimization. Thus, traditional neural network software systems with very slow hyperparameter optimization algorithms are not suitable for high-speed real-time learning and fast real-time prediction applications. In addition to the long training time problem, the conventional neural networks have the black-box problem (i.e., hyperparameters such as weights are not meaningful). How to build explainable open-box machine learning systems with low CO_2 emissions and low energy consumption is an important long-term problem.

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 [1]. Since CO_2e is proportional to the total computational power p_t: $CO_2e = 0.954p_t$ [2], effectively reducing training times results in greatly reducing both energy consumption and CO_2 emissions.

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.

Based on the successful implementation of the FPGA-based direct linear equation solver [3-5], 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. We will develop the novel Green Granular Convolutional Neural Network (GGCNN) 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.

2 A Building Block: an Efficient FPGA-based GGNN

2.1 Granular Sets

Different sets dealing with uncertainty of data and information, such as the fuzzy set [6], the neutrosophic fuzzy set [7], the intuitionistic fuzzy set [8], and Pythagorean fuzzy set [9], 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 $0 \leq \mu_A(x) \leq 1$, (2) $\nu_A(x)$ is degree of non-membership of x for $0 \leq \nu_A(x) \leq 1$, (3) $\phi_A(x)$ is certain degree of $\mu_A(x)$ for $0 \leq \phi_A(x) \leq 1$, (4) $\varphi_A(x)$ is uncertain degree of $\mu_A(x)$ for $0 \leq \varphi_A(x) \leq 1$, (5) $\theta_A(x)$ is certain degree of $\nu_A(x)$ for $0 \leq \theta_A(x) \leq 1$, and (6) $\vartheta_A(x)$ is uncertain degree of $\nu_A(x)$ for $0 \leq \vartheta_A(x) \leq 1$, where $0 \leq \mu_A(x) + \nu_A(x) \leq 1$, $0 \leq \phi_A(x) + \varphi_A(x) \leq 1$, and $0 \leq \theta_A(x) + \vartheta_A(x) \leq 1$.

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 $high$.

2.2 Software-FPGA Co-designed Learning

The green granular neural network (GGNN) with new fast software-FPGA co-designed learning iss designed using granular sets and the software-FPGA co-designed learning algorithm. 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.

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 ($a_{i1} \leq x_i \leq a_{i2}$, $a_{i2} \leq x_i \leq a_{i3}$, ..., $a_{i(m_i-1)} \leq x_i \leq a_{im_i}$). 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 [6]. 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 $M = 2^{n+1}$:

$$T_1^k q_1 + T_1^k q_2 + ... + T_M^k q_M = \psi_k \tag{1}$$

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 [3-5], 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.

We 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) [10, 11]. For instance, the new FPGA-based massive parallel data processing system can reduce CO_2 emissions by around 50% [11]. FPGA is a light-weight hardware with low CO_2 emissions and low energy consumption [12] 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 [3]. Thus, it is feasible to use FPGA to implement the new FPGA-based training algorithm.

3 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) [13].

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 [3, 14, 15], such as steepest descent method [16], QR decomposition [15] and Gauss Jordan method [17], 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 [18].

Furthermore, we used 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.

Table 1: Running times of the Python code and the FPGA code

Method	8	16	32	64	100
Python	1.139	1.581	7.057	30.147	76.581
FPGA	0.638	1.313	2.803	6.207	12.284

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

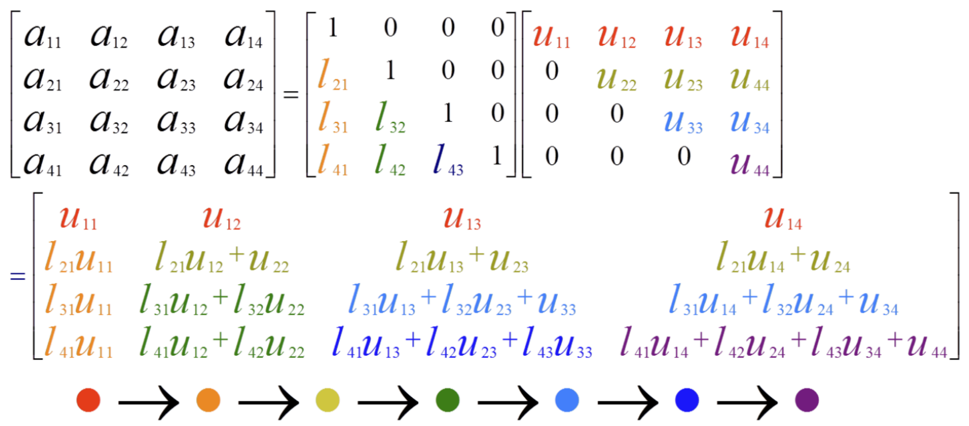

Figure 1: LU factorization on a 4×4 matrix [19]

consumption p_t for $CO_2e = 0.954p_t$ [2]. 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.

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.

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 [20-23] is given below:

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

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 2 to 4 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 2: 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

4 A New GGCNN with Software-FPGA Co-Designed Learning

Since the previous simulations indicate that the new software-FPGA-based learning method is feasible and useful to quickly train the GGNN, the GGNN can be used to build a new machine learning model as a basic building block. A CNN consists of convolutional layers, activation

Table 3: 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 4: 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

layers, pooling layers, and a classifier such as a MLP. The new GGCNN consists of convolutional layers, activation layers, pooling layers, a new FPGA-based GGNN layer called a FPGA learner, and a hybrid decision model for making a final decision. The new GGCNN with software-FPGA co-designed learning is shown in Fig. 3.

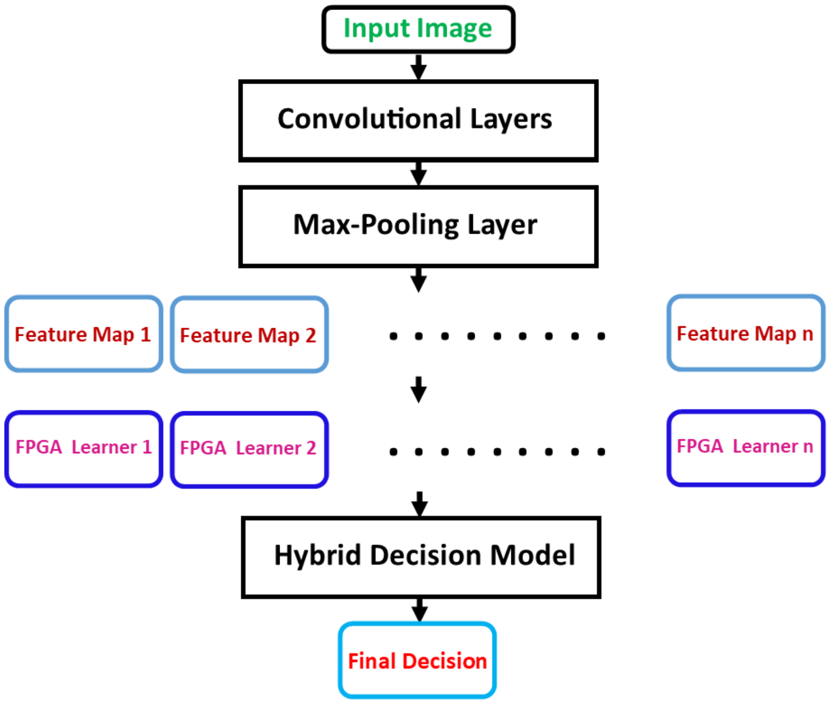

Figure 2: A GGCNN framework with Software-FPGA Co-Designed Learning

The FPGA-based direct hierarchical hyperparameter optimization algorithm for a GGCNN, a hybrid software-hardware-based algorithm, is given below as Algorithm 1. We assume that the linear equation solver can quickly solve a linear system of L hyperparameter equations. A $n \times n$

Feature Map (FM) has $n \times n$ features. N $n \times n$ feature maps FM_p for $p = 1, 2, ..., N$ are generated by the last pooling layer. $K = N \times n \times n$.

```
for $k = 1$ to $n$ do
    Step 1: Use software to partition $n \times n$ feature map $FM_p$ into $M_j$ $n_j \times n_j$ sub-feature
    maps for for $M_j = l_j \times l_j$, and $n_j < L$).
    Step 2: Use software to pre-calculate coefficients for $M_j$ linear systems of
    hyperparameter equations for $n_j \times n_j$ sub-feature maps): Use software to calculate
    coefficients such as $P_1^{1k}$, ..., $P_m^{1k}$, $P_1^{2k}$, . . ., $P_m^{2k}$ , $P_1^{3k}$, . . ., $P_m^{3k}$, and then calculate
    $P_1^{1k} = P_1^{1k} + S_1^{1k}$, ..., $P_m^{1k} = P_m^{1k} + S_m^{1k}$, $P_1^{2k} = P_1^{2k} + S_1^{2k}$, ..., $P_m^{2k} = P_m^{2k} + S_m^{2k}$,
    $P_1^{3k} = T_1^{3k} + S_1^{3k}$, ..., $P_m^{1k} = P_{2^n}^{3k} + S_m^{3k}$ of a linear system of hyperparameter equations.
    Step 3: Use the FPGA-based linear equation solver to solve $M_j$ linear systems of
    hyperparameter equations using $M_j$ $l \times l$ sub-feature maps): Use the FPGA-based linear
    equation solver to calculate optimal hyperparameters such as ($c_i$, $\eta_i$, and $\delta_i$) for
    $i = 1, 2, ..., m$ of each linear system of hyperparameter equations. The optimized
    hyperparameters are used to build new $M_j$ GGNNs with relevant granular knowledge
    bases with meaningful granular If-Then rules.
    Step 4: Use $M_j$ Use the $M_j$ FPGA-based GGNNs to make $M_j$ decisions $D_j^p$ for a new
    test feature map.
    Step 5: Use all individual decisions $D_j^p$ to make a hybrid decision.
end
```

Algorithm 1: Software-FPGA Co-Designed Learning Algorithm for the GGCNN

5 Conclusions

Initial simulation results indicates that the FPGA equation solver code ran faster than the Python equation solver code. In additon, the GGNN can 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. It is feasible to build the GGCNN using the GGNNs as basic building blocks to solve image recognition problems.

6 Future Works

In the future, the GGCNN 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 GGCNN models will be built.

A GGCNN 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 GGCNNs to solve the problem.

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 GGCNN.

After the software-FPGA co-designed learning is successful, a special high-speed FPGA hardware based direct linear equation solver as a fast learner will be implemented for building an efficient GGCNN with high classification accuracy to significantly reduce both CO_2 emissions and energy consumption.

References

[1] 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.

[2] 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.

[3] 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.

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

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

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

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

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

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

[10] Using FPGA chip system to reduce the carbon emissions of using web search (https://www.fpgakey.com/technology/details/using-fpga-chip-system-to-reduce-the-carbon-emissions-of-using-web-search).

[11] THALES LEVERAGES NEW TECHNOLOGIES TO BOOST BIOMETRIC MATCHING PERFORMANCE WHILST HAVING ENVIRONMENTAL IMPACT, 20 JAN 2020. (https://www.thalesgroup.com/en/group/journalist/press-release/thales-leverages-new-technologies-boost-biometric-matching).

[12] J. Morss, "FPGAs vs. GPUs: A Tale of Two Accelerators," January 16, 2019. https://www.dell.com/en-us/blog/fpgas-vs-gpus-tale-two-accelerators/

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

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

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

[23] 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.