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

Anonymous Author(s)
Affiliation
Address
e-mail

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 $A = \{ (x, \mu_A(x), \nu_A(x), \phi_A(x), \varphi_A(x), \theta_A(x), \vartheta_A(x)) : x \in X \}$, 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 $0 \leq \vartheta_A(x) \leq 1$.

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 is 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 $[a_{ij}, b_{ij}]$ 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 $k = 1, 2, ..., 2^n$. 

2
The granular rule base has $2^n$ granular IF-THEN rules for one database granule such that

\[ IF \ x_{i} \ is \ A_{1i}, \ and \ ... \ x_{n} \ is \ A_{ni}, \ THEN \ y \ is \ B_{k} \ for \ j_{i} \in \{1, 2, ..., n\}, \ and \ k = 1, 2, ..., 2^n. \]

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 $Q = \frac{1}{2} \sum_{k=1}^{K} [y_{k} - g(x_{1k}, x_{2k}, ..., x_{nk})]^2$. Based on $\frac{\partial Q}{\partial y_{j}} = 0$ for the GGNN, we can get a linear system of $M$-hyperparameter equations for $k = 1, 2, ..., M$.

\[
T_{1}^{k} y_{1} + T_{1}^{k} y_{2} + ... + T_{M}^{k} q_{M} = \psi_{k}
\]

(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 $4 \times 4$ 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>
<thead>
<tr>
<th>Method</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>1.139</td>
<td>1.581</td>
<td>7.057</td>
<td>30.147</td>
<td>76.581</td>
</tr>
<tr>
<td>FPGA</td>
<td>0.638</td>
<td>1.313</td>
<td>2.803</td>
<td>6.207</td>
<td>12.284</td>
</tr>
</tbody>
</table>

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 $CO_2 e = 0.954p_t$ [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 $f_k = (1 + x_0^0.5 + y_0^{-1} + z_0^{−1.5})^2$ is given below:

$$f_k = (1 + x_0^{0.5} + y_0^{-1} + z_0^{−1.5})^2.$$ (2)

The training data set with 8,000 training data is generated by $f_k$ shown in equation (2) such that $x_k^{tr} = 1.0 + \left\lfloor \frac{k}{400} \right\rfloor$, $y_k^{tr} = 1.0 + \left\lfloor \frac{k}{20} \right\rfloor$, $z_k^{tr} = 1.0 + k \mod 20$, where the operator mod is used, $f_k^{tr} \in [4.248, 55.833]$, and $k = 0, 1, ..., 7,999$. A testing data set with 6,859 testing data is generated by $f_k$ such that $x_j^{te} = 1.5 + \left\lfloor \frac{j}{361} \right\rfloor$, $y_j^{te} = 1.5 + \left\lfloor \frac{j}{19} \right\rfloor$, $z_j^{te} = 1.5 + j \mod 19$, 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>
<thead>
<tr>
<th>Neural Network</th>
<th>MSE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-Layer ANN</td>
<td>58.22</td>
<td>7.63</td>
</tr>
<tr>
<td>20-Layer ANN</td>
<td>63.44</td>
<td>6.88</td>
</tr>
<tr>
<td>GGNN</td>
<td>47.31</td>
<td>6.88</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Neural Network</th>
<th>MSE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-Layer ANN</td>
<td>55.68</td>
<td>7.46</td>
</tr>
<tr>
<td>20-Layer ANN</td>
<td>76.99</td>
<td>8.77</td>
</tr>
<tr>
<td>GGNN</td>
<td>46.38</td>
<td>6.81</td>
</tr>
</tbody>
</table>

Table 4: Function Prediction Performance Comparison between ANNs and the GGNN for $f^1$ for 1,000 training epochs.

<table>
<thead>
<tr>
<th>Neural Network</th>
<th>MSE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-Layer ANN</td>
<td>51.16</td>
<td>7.15</td>
</tr>
<tr>
<td>20-Layer ANN</td>
<td>53.66</td>
<td>7.33</td>
</tr>
<tr>
<td>GGNN</td>
<td>46.71</td>
<td>6.83</td>
</tr>
</tbody>
</table>

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.

```
158 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 \( p = 1, 2, \ldots, N \) are generated by the last pooling layer. \( K = N \times n \times n \).

For \( k = 1 \) to \( n \) do

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 \( n_j < L \).
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 \( p_{1k}, \ldots, p_{m1}, p_{2k}, \ldots, p_{m2}, p_{3k}, \ldots, p_{mn} \), and then calculate \( p_{1k} = p_{1k} + s_{1k}, \ldots, p_{m1} = p_{m1} + s_{1k}, p_{2k} = p_{2k} + s_{2k}, \ldots, p_{mn} = p_{mn} + s_{2k} \), \( p_{3k} = T_{3k}^{t_{3k}} + s_{3k}^{t_{3k}}, \ldots, p_{3k}^{n} = T_{3k}^{t_{3k}} + s_{3k}^{n} \) of a linear system of hyperparameter equations.
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 \( (c_i, \eta_i, \text{ and } \delta_i) \) for \( i = 1, 2, \ldots, m \) 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.
4. Use \( M_j \) Use the \( M_j \) FPGA-based GGNNs to make \( M_j \) decisions \( D_p \) for a new test feature map.
5. Use all individual decisions \( D_p \) to make a hybrid decision.

end

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

5 Conclusions

Initial simulation results indicate that the FPGA equation solver code ran faster than the Python equation solver code. In addition, 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 GGNN 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 GGNNs 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


