IPSJ Transactions on System LSI Design Methodology Vol.13 69–71 (Aug. 2020)

[DOI: 10.2197/ipsjtsldm.13.69]

Short Paper

R-GCN Based Function Inference for Gate-level Netlist

Motoki Amagasaki[1,a)] Hiroki Oyama[1] Yuichiro Fujishiro[1] Masahiro Iida[1]
Hiroaki Yasuda[†1] Hiroto Ito[†1]

Received: December 4, 2019, Revised: March 11, 2020,
Accepted: April 23, 2020

Abstract: Graph neural networks are a type of deep-learning model for classification of graph domains. To infer arithmetic functions in a netlist, we applied relational graph convolutional networks (R-GCN), which can directly treat relations between nodes and edges. However, because original R-GCN supports only for node level labeling, it cannot be directly used to infer set of functions in a netlist. In this paper, by considering the distribution of labels for each node, we show a R-GCN based function inference method and data augmentation technique for netlist having multiple functions. According to our result, 91.4% accuracy is obtained from 1,000 training data, thus demonstrating that R-GCN-based methods can be effective for graphs with multiple functions.

Keywords: graph neural network, R-GCN, function inference

1. Introduction

Recently, many studies have investigated extension of deep learning methods to graph data. Graph neural networks (GNNs) [1] are a type of learning model for classification of graph domains. A typical GNN model is graph convolutional networks (GCNs), which apply convolutional operations to graphs. In Refs. [2], [3], a semi-supervised learning GCN is used to categorize a citation dataset. In these graphs, research papers are treated as nodes and citation relations as edges. However, because GCN performs a convolutional operation based on the graph Fourier transform, only an undirected graph can be used as input.

To eliminate such restrictions on graph convolution, Refs. [4], [5] proposed relational graph convolutional networks (R-GCN), which can directly treat relations between nodes and edges. R-GCN convolves input and output relations with neighboring nodes by defining a graph convolutional operation using only connection information for nodes and edges. This allows handling complex relations between nodes such as directed graphs, self-loops, and multiple edges that were not possible with previous GCNs.

In this study, we focus on the graph classification ability of R-GCN and propose a function inference method for a gate-level netlist described in Verilog Hardware design language (HDL). Originally, specification sheets and RTLs are usually managed together in SoC design. However, RTL does not exist when designed directly at the schematic or gate level. These are problems if designers want to know the functions of these circuits. In addition, if the gate-level netlist is large, it is not easy for designer to know the functions. For these reason, in order to mitigate these problems, we proposed a method to estimate a known function sets (addition, subtraction, multiplication, and multiplexer) for a gate-level netlist. Our goal is to infer functions when specifications for the gate-level netlist are unclear. However, since R-GCN only labels each node, it cannot be used to infer function sets in a netlist. To solve this problem, we propose a classification method and data augmentation technique for graphs having multiple functions.

[1] Graduate School of Science and Technology, Kumamoto University, Kumamoto 860–8555, Japan
[†1] Presently with Mitsubishi Electric Engineering Company Limited
[a)] amagasaki@cs.kumamoto-u.ac.jp

2. Function Inference based on R-GCN

2.1 Problem Definition

The target netlist has an arithmetic function composed of adder, subtractor, multiplier, and multiplexer (MUX) functions. These are mapped in standard cells, and each function has a hierarchy with an individual instance. For example, a netlist composed of 7 functions has 7 instances.

2.2 Network Structure

R-GCN defines a graph-convolutional operation based on only the connection relations between nodes and edges in Eq. (1). R-GCN can thus treat more complex graphs, such as directed graphs, multiple edges, and self-loops.

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in N_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right) \quad (1)$$

In Eq. (1), $h_i^{(l+1)}$ is the state of node i in hidden layer $(l+1)$ of the neural network. R denotes the set of relations in the graph, N_i^r denotes the set of neighbor indices of node i under relation $r \in R$, and $W_r^{(l)}$ denotes the weight matrix of neighboring nodes with relation r to node i in hidden layer l of the neural network. σ is an element-wise activation function. $C_{i,r}$ is a normalization constant indicating the total number of edges for each relation. The relation indicates the edge attribute. Equation (1) takes a lin-

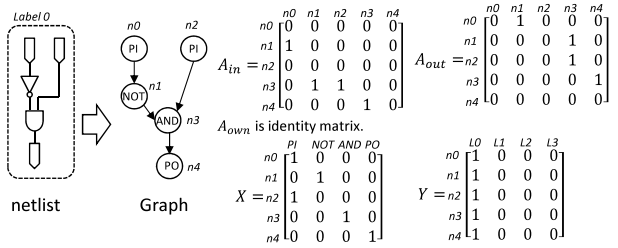

Fig. 1 Input matrix for R-GCN.

ear sum of all neighboring nodes of node i and then adds its state ${W_0}^{(l)}{h_i}^{(l)}$. **Figure 1** shows an example R-GCN matrix. The relations are defined as input side A_{in}, output side A_{out}, and self A_{own}. Size of Matrix A_{in}, A_{out}, A_{own} are *the number of nodes in netlist* $\times$ *the number of nodes in netlist*, respectively. Feature matrix X has elements for the type of gate cell, and feature of each node is expressed as a one-hot vector with the corresponding gate as 1. Thus, size of Matrix X is *the number of nodes in netlist* $\times$ *type of gate cell*. R-GCN realizes convolution of nodes considering these relations. When calculating 1st hidden layer, matrix h in Eq. (1) can be replaced with matrix X. $r \in R$ and $j \in {N_i}^r$ in left term are denoted relationship of A_{in}, A_{out}. Because A_{own} is identity matrix, there is ${W_0}^{(l)}{h_i}^{(l)}$ only in right term.

In this paper, the standard cell and primary input–output in a netlist are represented as a node and its connection as an edge. However, because R-GCN performs labeling for each node, the netlist function cannot be estimated. The netlist we use is instantiated by functional unit. Therefore, each function is determined by majority vote between nodes of the same instance.

2.3 Training and Data Augmentation

To perform high accuracy and obtain generalized performance, we modify the calculation of the loss function during learning. The loss in our R-GCN uses cross-entropy error $E = -\frac{1}{N}\sum_{k=1}^{N} t_k \log z_k$. N is the total number of nodes in the netlist, k is the current node, t_k is the label data of node k, and z_k is the inference result for each node k.

There are multiple functions in the netlist, so it is not enough to learn only individual functions in the training phase. Therefore, to increase precision for multiple functions, data augmentation is performed by adding various functions to the input and output parts of target function to be trained. In training with multiple functions, loss is calculated by focusing on only nodes in the target function. For example, added functions for data augmentation are not subject to loss calculations. As a result, we expect that accurate inference is performed for nodes located at boundaries between functions.

3. Evaluation

3.1 Evaluation Method

R-GCN based function inference is performed for netlists with adder, subtractor, multiplier, or MUX functions. We use a netlist with 1 to 4 functions as learning data and evaluate the classification success rate of netlists with at most 7 functions as test data. Success is defined as more than half of the nodes in an instance being correctly classified. When there are multiple functions, classification is considered successful only if all functions in the netlist are correctly classified. We prepared ripple-carry adder, carry look-ahead adder, and carry-select adder circuits for addition and subtraction. Multiplier circuits use array-type and Wallace tree–type algorithms.

Table 1 Dataset for evaluations.

	Training data	Validation data	Test data
Number of data	631	631	1,000

Table 2 Hyperparameters.

Parameters	Value
Epoch	200
Early stopping	20
Number of standard cells	127
Dropout rate	0.655492692944
Optimization algorithm	Ndam
Learning rate	0.000243964937345
Number of hidden layers	5
Neurons in hidden layers	135
Activation function	ReLU

Table 3 Evaluation results.

	Success	Failure	Accuracy (%)
Test data	914	86	91.4

Table 4 Average success rates.

Classification result	Success rate (%)
Success	96.7
Failure	10.3

3.2 Evaluation Conditions

Data circuits are prepared in gate-level HDL synthesized by Synopsis Design Compiler for area- and delay-oriented optimization. **Table 1** shows numbers of data used for R-GCN training and evaluation. The netlist used for learning and verification includes at most 4 functions, and created a combination of all adder, subtractor, multiplier, and multiplexer patterns. Test data were 1,000 netlists with at most 7 functions and random combinations of features. To optimize hyperparameters for learning R-GCN, we used the automatic parameter optimization tool Optuna [6]. **Table 2** shows the hyperparameters used. We used Keras 1.2.1 as the learning framework, and created the R-GCN input matrix from the gate-level HDL using a custom-developed converter, "logic2vec" [7].

3.3 Evaluation Results

Table 3 shows classification results and success rates for netlists with at most 7 functions. We found approximately 91.4% accuracy, as shown in Table 3. **Table 4** shows average classification success rates for each success and failure pattern. In "Success" and "Failure" in Table 3, the node ratio that showed the correct answer per function was calculated, and the average value of validation data was defined as "the average success ratio". From Table 4, we can see that 96.7% of nodes determined the correct label for successful function. In contrast, only 10.3% of failing functions were classified as successful. The circled part in **Fig. 2** represents individual instance parts, and the color of each node shows the classification result. Most of the misclassified netlists were due to the misclassification of MUX as multiplier. In other

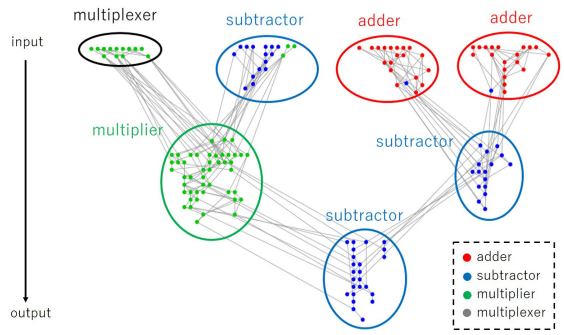

Fig. 2 Example of failed classification for 7 functions.

cases, adders were misclassified as subtractors. This occurred because the MUX had few nodes, and was therefore misjudged as a part of the neighboring multipliers. Also, subtractors and adders have highly similar structures. Figure 2 shows actual results for netlists that failed to be classified.

4. Conclusion

We obtained 91.4% accuracy in function inference using R-GCN for gate-level netlists with multiple functions. In future work, we will stratify inferred functions and experiments using large application level inference.

References

[1] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P.S.: A Comprehensive Survey on Graph Neural Networks, arXiv preprint arXiv:1901.00596 (Aug. 2019).
[2] Kipf, T.N. and Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks, *Proc. International Conference on Learning Representations* (*ICLR*) (Feb. 2017).
[3] Chen, Z., Kolhe, G., Rafatirad, S., Homayoun, H., Zhao, L. and Lu, C.T.: Estimating the Circuit Deobfuscating Runtime based on Graph Deep Learning, arXiv preprint arXiv:1902.05357 (Feb. 2019).
[4] Schlichtkrull, M., Kipf, T.N., Bloem, P., Berg, R., Titov, I. and Welling, M.: Modeling Relational Data with Graph Convolutional Networks, arXiv preprint arXiv:1703.06103 (Mar. 2017).
[5] Thomas Kipf: Keras-based implementation of Relational Graph Convolutional Networks, available from ⟨https://github.com/tkipf/relational-gcn⟩ (accessed 2019-10).
[6] Preferred Networks, Inc.: Optuna - A hyperparameter optimization framework, available from ⟨https://optuna.org/⟩ (accessed 2019-10).
[7] Fujishiro, Y., Oyama, D., Amagasaki, M., Iida, M., Yasuda, T. and Ito, H.: Gate Level Netlist Function Classification Method Based on R-GCN, IEICE Technical Report, VLD2019-30, Vol.119, No.282, pp.7–12 (Nov. 2019) [In Japanese].

Motoki Amagasaki received his B.E. and M.E., degrees in Control Engineering and Science from Kyushu Institute of Technology, Japan in 2000, 2002, respectively. He was a engineer at NEC Micro Systems Co., Ltd. from 2002–2005. He received his D.E. degree from Kumamoto University, Japan, in 2007. He has been a associate professor in the Faculty of Advanced Science and Technology at Kumamoto University since 2019. His research interests machine learning, reconfigurable system and VLSI design. He is a member of IEICE, IPSJ and IEEE.

Hiroki Oyama was received his B.E. degree in Computer Science from Kumamoto University in 2018. Further, he received his M.E. degree in Computer Science and Electrical Engineering from Kumamoto University in 2020.

Yuichiro Fujishiro was received his B.E. degree in Computer Science from Kumamoto University in 2019.

Masahiro Iida received his B.E. degree in Electronic Engineering from Tokyo Denki University in 1988. He was a research engineer at Mitsubishi Electric Engineering Co., Ltd. from 1988 to 2003. He received his M.E. degree in Computer Science from Kyushu Institute of Technology in 1997. Further, he received his D.E. degree from Kumamoto University, Japan, in 2002. He was an associate professor at Kumamoto University until 2015, and during 2002–2005, he held an additional post as a researcher at PRESTO, Japan Science and Technology Corporation (JST). He has been a professor in the Faculty of Advanced Science and Technology at Kumamoto University since January 2016. His current research interests include high-performance low-power computer architectures, Neural Network Accelerator, FPGA computing, VLSI devices and design methodology. He is a senior member of the IPSJ and the IEICE, and a member of IEEE.

Hiroaki Yasuda received his M.E. degree from Aichi Institute Technology in 2010. He joined MITSUBISHI ELECTRIC ENGINEERING CO., LTD. in 2010, and is now a engineer in Factory Automation LSI Engineering Section.

Hiroto Ito received his B.E. degree from Meijo University in 2008. He joined MITSUBISHI ELECTRIC ENGINEERING CO., LTD. in 2008, and is now a engineer in Factory Automation LSI Engineering Section.

(Recommended by Associate Editor: *Nozomu Togawa*)