Retraining-Based Timing Error Mitigation for Hardware Neural Networks

Jiachao Deng[*,†], Yuntan Fang[*,☆,‡], Zidong Du[*,†], Ying Wang[*], Huawei Li[*],
Olivier Temam[§], Paolo Ienne[‡], David Novo[‡], Xiaowei Li[*], Yunji Chen[*,✧], Chengyong Wu[*]
[*]SKL Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China,
[†]University of Chinese Academy of Sciences, Beijing, China,
[☆]Shannon Laboratory, Huawei Technologies Co., Ltd., China, [§]Google Inc., CA, USA,
[‡]École Polytechnique Fédérale de Lausanne (EPFL), Switzerland,
[✧] Center for Excellence in Brain Science, Chinese Academy of Sciences, Beijing, China,
{dengjiachao, duzidong, wangying2009, lihuawei, lxw, cyj, cwu}@ict.ac.cn, fangyuntan@huawei.com,
olivier.temam@google.com, {paolo.ienne, david.novobruna}@epfl.ch

Abstract **- Recently, neural network (NN) accelerators are gaining popularity as part of future heterogeneous multi-core architectures due to their broad application scope and excellent energy efficiency. Additionally, since neural networks can be retrained, they are inherently resillient to errors and noises. Prior work has utilized the error tolerance feature to design approximate neural network circuits or tolerate logical faults. However, besides high-level faults or noises, timing errors induced by delay faults, process variations, aging, etc. are dominating the reliability of NN accelerator under nanoscale manufacturing process. In this paper, we leverage the error resiliency of neural network to mitigate timing errors in NN accelerators. Specifically, when timing errors significantly affect the output results, we propose to retrain the accelerators to update their weights, thus circumventing critical timing errors. Experimental results show that timing errors in NN accelerators can be well tamed for different applications.**

Keywords-neural networks; error tolerance; machine learning; timing errors; overclocking

I. INTRODUCTION

Driven by dark silicon [1], modern computer architectures are evolving towards heterogeneous multi-core platforms mixed with cores and accelerators like GPUs, FPGAs and ASICs. Recently, researchers have explored hardware neural networks (HNNs) as promising accelerators, which achieve a good balance between energy efficiency and application scope. It is demonstrated that neural network accelerators significantly reduce energy consumption [2] and at the same time exhibit a broad application scope [3]. Specifically, neural networks can well accommodate the emerging high-performance machine learning applications such as recognition, mining, and synthesis (RMS) [4].

Another feature of neural networks is their inherent tolerance to errors. In the light of the error sources, errors occurring in neural networks can be voluntary or involuntary. On one hand, in order to achieve the benefit of energy efficiency, errors are voluntarily introduced into the computation units of neural networks, i.e., the circuit is implemented approximately at design time, such as in [5] and [7]. On the other hand, due to the non-ideal manufacturing process, errors can arise because of involuntary faults. Research efforts have shown that software neural networks are tolerant to permanent faults in GPUs [8] and a custom hardware neural network is tolerant to transistor-level defects [2]. The root cause of the error tolerance capability of neural networks lies in that they can be retrained.

Although prior work has investigated the error tolerance property of neural networks, the context is limited to voluntary logical errors at design time and involuntary logical faults during the manufacturing process, both of which change the circuit structure. However, timing errors are a missing part in the research domain of neural networks. As a matter of fact, timing errors are another important source that affects the normal operation of circuits. Significantly different from logical errors, timing errors change the timing behavior rather than the structure of circuits. Therefore, it is of great value to investigate how timing errors will affect the operation of HNNs and whether the errors can be mitigated by leveraging the intrinsic retraining feature of neural networks.

Timing errors are becoming more and more dominant with continuous technology scaling. The sources of timings errors are various, including delay defects, process variations, power supply noise, crosstalk, and aging. Additionally, in order to gain the energy or performance benefit, voltage overcalling or frequency overclocking can also lead to timing errors. When timing errors occur in the NN accelerators, the mean squared error (MSE) will soar and the accuracy will decrease, which means the NN accelerators work incorrectly as shown in our experiment results.

In this paper, *we mainly focus on the impacts of timing errors on HNNs and propose a timing variation-aware retraining method for HNNs to suit the de-facto distribution of timing variation in each individual chip, thereby mitigating the negative effects of serious timing violation through the intrinsic resilience of neural networks, which constitute the major contribution of this paper*. Our results show that the impacts of timing errors on neural networks can be well alleviated by ad-hoc retraining.

The rest of this paper is organized as follows. In Section II, we introduce MLP-based neural networks and timing error modeling. Section III presents the experimental methodology and Section IV shows the experimental setups and results. Related work is introduced in Section V and conclusions are drawn in Section VI.

This work was supported in part by National Natural Science Foundation of China (NSFC) under grant No. (61432017, 61176040, 61221062), and in part by National Basic Research Program of China (973) under grant No. 2011CB302501.

II. MLPs and Timing Error Modeling

Although we focus on MLPs in this paper, the concepts and methodologies on timing error mitigation are also applicable to other types of NNs, such as DNNs [6].

A. Multi-Layer Perceptrons (MLPs)

MLPs are widely used neural networks. Figure 1 [5] shows a 2-layer MLP, i.e., in addition to the default input layer, it is composed of a hidden layer and an output layer. Each synapse in the layers implements a multiplication of the weight and the neuron output from the previous layer. Each neuron first accumulates the results of all its synapses and then applies a sigmoid transform to the sum, where the sigmoid activation function is implemented through piece-wise linear approximation. Overall, a hardware MLP neural network is a computation-intensive circuit where multiplications and additions are dominant. More details about MLPs can be found in [5].

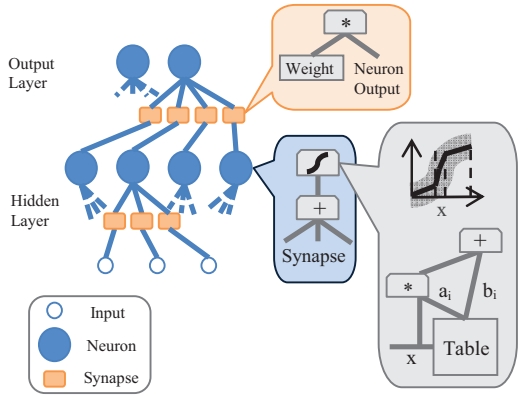

Figure 1. Structure and logic operators of a 2-layer MLP [5].

B. Timing Error Modeling

In this paper, we are interested in the behavior of HNNs in the situation of low-power or near-threshold operation. Typically, for such cases, the clock frequency will be aggressively set while the supply voltage will be kept low to achieve higher energy efficiency. Therefore, if the operating frequency of overclocking HNNs is close to the safe critical point, the timing error will occur due to insufficient timing margin. Without loss of generality, we model the timing errors in the MLP circuit shown in Figure 1 as follows. After obtaining the gate-level netlist of the MLP neural network, we introduce extra delay variation into each gate. For each gate, we inject the same ratio of extra delay relative to the nominal gate delay, simulating the voltage overscaling or overclocking situation. For example, if a 10% delay variation is injected into a gate with a 50ps typical delay, then the gate delay becomes 55ps. Delay variation is not injected on the wires, but we still consider the delay of all wires during the timing-aware simulation process. Once the accumulated delay of all gates and wires along a path exceeds the specified clock cycle, a timing violation occurs. At this point, if the captured result during simulation is different from the expected output, there exists a timing error in the circuit.

III. Methodology of Timing Variation Aware Retraining

In this section, we describe the overall methodology of timing variation aware retraining.

A. Overall Methodology

Figure 2 shows the overall methodology flow. In this paper, an MLP-based neural network is modeled at register-transfer level (RTL) and has three pipeline stages, i.e., one stage for input layer, one stage for the hidden layer and the last stage for the output layer. Accurate timing information of the circuit can be obtained using three steps. Firstly, we synthesize the circuit from RTL into a gate-level netlist using synthesis tools with technology library under Standard Delay Constraints Files which mainly defines the clock period and I/O delay. Secondly, we place and route the netlist. And thirdly, we conduct static timing analysis (STA) to get the timing information of the neural network circuit. After these steps, timing aware gate-level simulation could uncover the behavior of the neural network circuits under a given working clock frequency. Then, forward propagation is conducted on the target HNN circuit using the datasets of real-world applications as inputs under the function clock of the circuit. After that, classification mean squared error (MSE) calculation is performed on the circuit outputs data obtained from forward propagation. The current MSE is compared with the previous minimum MSE. If it is smaller than the minimum MSE, we will update the minimum MSE and record current accuracy and weights. At last, the backward propagation algorithm will be used to generate modified weights to be fed back to the HNN for retraining. Retraining iterates until the maxmum epoch is reached.

Various datasets of machine learning applications are used as testbench inputs of the neural network circuit, which we will introduce in Section III-B. Since we have no real HNN at hands, we inject timing variation in the netlist of 2-layer MLP to stimulate timing errors as discussed in Section II-B by modifying the timing file obtained from STA tools. So forward propagation results, i.e., the outputs generated from the circuit, are collected using the gate-level timing simulator with physical layout information, as demonstrated in Figure 2.

Subsequently, the outputs generated from the circuit are collected to train the neural network. The most popular back propagation (BP) training algorithm [9] is utilized. The neural network is trained on each benchmark respectively using 10-fold cross-validation, and the initial weights of the neural network are randomly set with a uniform distribution. We record the minimum MSE of testing and the corresponding classification accuracy and neuron weights. The whole simulation procedure iterates until the maximum epoch number is reached. Finally, the above training process is repeated for 10 times under each timing configuration and we choose the training result with the best MSE of testing among them.

B. Benchmarks and Evaluation Metrics

To evaluate the effects of timing errors on HNNs, we use 5 benchmark applications from the UCI machine learning repository [10], they are *iris, wine, ionosphere, glass and sonar*. We use two metrics to evaluate the results of neural networks with and without timing errors. The first metric is the mean squared error (MSE) between desired outputs and actual outputs, which reflects the average classification error. The other is the practical classification accuracy, i.e., the percentage of correct classification. In the ideal case for

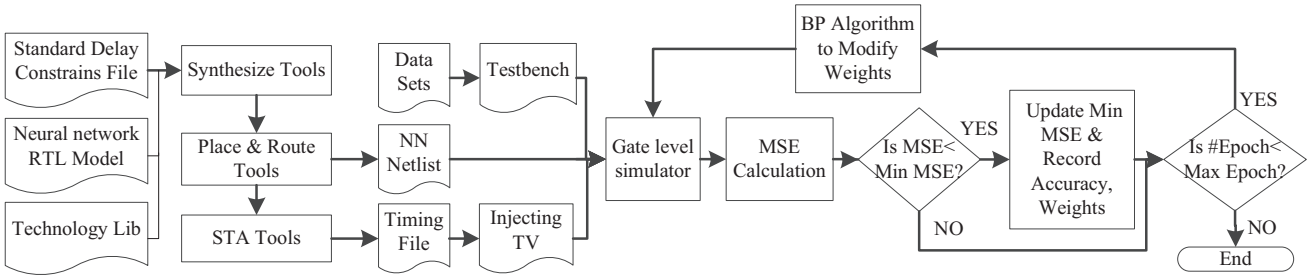

Figure 2. Overall methodology flow.

neural networks, MSE should be as small as possible and classification accuracy should be as high as possible.

In sum, the overall experimental flow in this paper is based on accurate timing aware gate-level simulation, which could uncover the real behavior of the neural network circuit. What's more, using the datasets of real-world applications as inputs can better exercise the function of the neural network circuit.

IV. Experimental Setups And Results

In this section, we present the experimental setups and evaluation results of the MLP-based neural network circuit under various benchmark applications and timing variation rates. More specifically, we compare the results of the baseline, the pre-retraining case and post-retraining case. Here, the baseline is free from timing variations in the circuit, and before retraining case and after retraining case indicate the baseline-identical HNNs in presence of timing variation and the HNNs with new synaptic weight configuration after retraining, respectively.

A. Experimental Setups

We choose the state-of-the-art EDA tools to conduct experiments. We use Synopsys Design Compiler to do synthesis, use IC Compiler to place & route, and use Prime Time to do STA, which is the standard back-end flow of IC design. Synopsys VCS is used as gate-level simulator.

We inject the timing variation in Standard Delay Files (SDF) which is produced by STA tools PrimeTime. As an IEEE standard for the representation and interpretation of timing data, SDF contains delay information of both gates and wires.

In order to accommodate a large number of applications and provide good classification results, the numbers of its inputs, hidden neurons and output neurons are set as 90, 10, and 10, respectively [5]. Of course, we could configure the numbers of inputs, hidden and output neurons based on different applications, for example, for the data set of *sonar*, we use 60 input, 10 hidden neurons and 2 output neurons.

To choose the NN parameters that result in the lowest MSE and highest accuracy, we explored all of combination space of the parameters *#Hidden* and learning rate η of every benchmark. As shown in Table I, the first column is the name of applications. The second column describes the function of each application. The third to fifth columns represent the number of inputs, the number of hidden neurons, and the number of output neurons the application uses, respectively. The last two columns show the learning rate (η) and the epoch number, respectively. As a trade-off between the training time and training quality, the epoch number is empirically adjusted according to the used application.

TABLE I. Benchmark Characteristics

Application	Function Description	#In	#Hidden	#Out	Learning rate η	#Epoch
iris	Plants classification	3	8	3	0.10	300
wine	Wine origin based on chemicals	13	2	3	0.10	300
ionosphere	Radar returns from ionosphere	34	6	2	0.10	150
glass	Glass oxides identification (forensic)	9	10	6	0.05	300
sonar	Metal vs. Rock sonar returns	60	10	2	0.05	200

B. Experimental Results

Each subfigure of Figure 3 shows the normal behavior of the baseline neural network circuit (no timing variation) and the effects of timing variations on the neural network circuit before retraining and after retraining under each application provided in Table I. In our experiments, the injected extra delay of each gate varies from 10% to 40% to generate 10 different timing files for simulation. For the baseline free from timing variation, MSE is relatively small and negligible. With the timing variation worsening, as shown in Figure 3, the overall MSE grows larger under all applications before and after retraining. However, compared to the result before retraining, MSE after retraining is significantly reduced, which means that the error induced by timing variation is compensated or mitigated by retraining. Figure 4 and Figure 5 reveal the average MSE and classification accuracy under all applications, respectively. From Figure 4, we can observe that the average MSE is reduced after retraining. When the injected timing variation doesn't exceed 30%, the average MSE after retraining is close to that of the baseline. Correspondingly, the average classification accuracy shown in Figure 5 is improved by retraining. For example, when the range of timing variation is 30%, the average MSE before retraining is 0.61 while that after retraining is reduced to 0.21. Accordingly, the average classification accuracy before retraining is by 44% while that after retraining is improved to 76%.

As demonstrated above, timing errors can be well tamed by retraining the neural network circuit. The root cause lies in that retraining changes the values of weight and bias parameters, thus partly circumventing problematic paths that

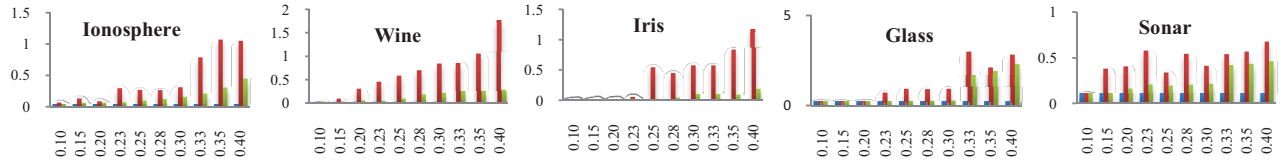

Figure 3. Effects of timing variations on MSE. Baseline(blue), before retraining(red) and after retraining(green).

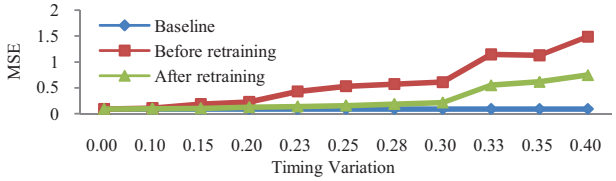

Figure 4. Effects of timing variations on average MSE.

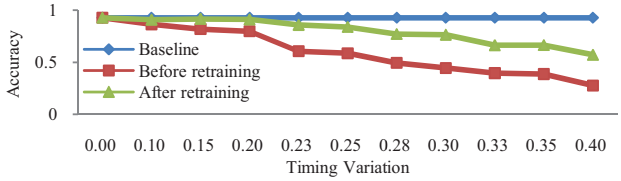

Figure 5. Effects of timing variations on average classification accuracy.

generate serious errors at the outputs. It is the intrinsic retraining capability of neural networks that enables the strong tolerance to timing errors.

V. Related Work

The most relevant work with this paper mainly falls into three areas, i.e., (a) HNNs, (b) approximate design and (c) error tolerance-aware circuit testing.

Hardware neural networks. HNNs are emerging as a kind of promising accelerator and researchers have investigated their features from various aspects. HNN accelerators have a broad application scope [3] and energy efficiency [5, 7], and have better performance and energy efficiency in the analog scheme [11]. Additionally, the fault tolerance of neural networks to logical faults has been examined in GPUs [8] and a custom design [2]. Nevertheless, timing errors have never been considered in previous work on neural network circuits.

Approximate design. By leveraging the characteristics of error-tolerant applications, approximate computing is an effective paradigm to trade accuracy for other benefits such as energy efficiency and performance. In addition to the above mentioned approximation techniques used in neural networks, there are extensive other approximate design methods at the circuit, architecture and algorithm level, for example, approximate storage has been used to phase change memory in multimedia applications [12] and general-purpose computing platforms [13].

Error tolerance-aware circuit testing. The circuit testing community has explored various testing techniques for error-tolerant applications. The core idea is to identify uncritical faults and only test the critical faults, thereby improving the effective yield of chips. The work in [14] proposes a cross-layer fault criticality evaluation framework. Circuit-level testing techniques based on error rate or error significance have been proposed in [15, 16].

Significantly different from the previous work, this paper targets HNN accelerators with a much broader application scope relative to ASICs like video codecs, focuses on timing errors within them and retrain each neural network chip in an ad hoc way to mitigate timing errors.

VI. Conclusions

HNN accelerators are becoming more and more popular in the dark silicon era. However, the capability of their tolerance to timing errors has not been investigated before. In this paper, we model the propagation of timing errors in HNNs and explore its implications on them. Furthermore, we leverage the incremental retraining ability of neural networks to mitigate timing errors. Our evaluation results show that retraining HNNs can effectively alleviate timing errors.

In the future work, we will study the effects of timing errors on other types of neural networks and leverage their timing error tolerance to guide the design of high-performance and energy-efficient neural networks.

References

[1] H. Esmaeilzadeh, E. Blem, R. S. Amant, et al., "Dark silicon and the end of multicore scaling," in *ISCA*, pp. 365-376, 2011.
[2] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *ISCA*, pp. 356-367, 2012.
[3] T. Chen, Y. Chen, M. Duranton, et al., "BenchNN: On the broad potential application scope of hardware neural network accelerators," in *IISWC*, pp. 36-45, 2012.
[4] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," Technology@Intel Magazine, pp. 1-10, Feb. 2005.
[5] Z. Du, A. Lingamneni, Y. Chen, et al., "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *ASP-DAC*, pp. 201-206, Jan. 2014.
[6] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ASPLOS*, pp.269-283, March 2014.
[7] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems," in *ICCAD*, pp. 130-137, 2013.
[8] A. Hashmi, H. Berry, O. Temam, et al., "Automatic abstraction and fault tolerance in cortical microarchitectures," in *ISCA*, pp. 1-10, 2011.
[9] S. Haykin, Neural networks. Prentice Hall Intl, London, UK, 2nd edition, 1999.
[10] A. Asuncion and D. J. Newman, "UCI machine learning repository," http://archive.ics.uci.edu/ml/
[11] R. S. Amant, et al., "General-purpose code acceleration with limited-precision analog computation," in *ISCA*, 2014.
[12] Y. Fang, H. Li, and X. Li, "Lifetime enhancement techniques for PCM-based image buffer in multimedia applications," *IEEE TVLSI*, vol. 22, no. 6, pp. 1450-1455, June 2014.
[13] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *MICRO*, pp. 25-36, 2013.
[14] Y. Fang, H. Li, and X. Li, "A fault criticality evaluation framework of digital systems for error tolerant video applications," in *ATS*, pp. 329-334, 2011.
[15] K.-J. Lee, T.-Y. Hsieh, and M. A. Breuer, "Efficient over-detection elimination of acceptable faults for yield improvement," *IEEE TCAD*, vol. 31, no. 5, pp. 754-764, May 2012.
[16] Z. Jiang et al., "Threshold testing: improving yield for nanoscale VLSI," *IEEE TCAD*, vol. 28, no. 12, pp. 1883-1895, Dec. 2009.