An Assessment of Vulnerability of Hardware Neural Networks to Dynamic Voltage and Temperature Variations

Xun Jiao[‡], Mulong Luo[§], Jeng-Hau Lin[‡], and Rajesh K. Gupta[‡]
[‡]Department of Computer Science and Engineering, UC San Diego, CA, USA
[§]Department of Electrical and Computer Engineering, Cornell University, NY, USA
{xujiao,jel252,gupta}@cs.ucsd.edu, ml2558@cornell.edu

***Abstract*—As a problem solving method, neural networks have shown broad applicability from medical applications, speech recognition, and natural language processing. This success has even led to implementation of neural network algorithms into hardware. In this paper, we explore two questions: (a) to what extent microelectronic variations affects the quality of results by neural networks; and (b) if the answer to first question represents an opportunity to optimize the implementation of neural network algorithms. Regarding first question, variations are now increasingly common in aggressive process nodes and typically manifest as an increased frequency of timing errors. Combating variations – due to process and/or operating conditions – usually results in increased guardbands in circuit and architectural design, thus reducing the gains from process technology advances. Given the inherent resilience of neural networks due to adaptation of their learning parameters, one would expect the quality of results produced by neural networks to be relatively insensitive to the rising timing error rates caused by increased variations. On the contrary, using two frequently used neural networks (MLP and CNN), our results show that variations can significantly affect the inference accuracy. This paper outlines our assessment methodology and use of a cross-layer evaluation approach that extracts hardware-level errors from twenty different operating conditions and then inject such errors back to the software layer in an attempt to answer the second question posed above.**

I. Introduction

Neural network algorithms have found use in a wide range of applications such as medical diagnostics [26], image classification [19], speech recognition [12], and natural language processing [6]. This versatility has led to their implementation on a variety of hardware platforms: GPU [5], FPGA [11], and ASIC [4].

With the continuous scaling of CMOS technology, the underlying transistors in all these implementations are increasingly susceptible to variations in manufacturing and operating conditions. Dynamic variations in microelectronic systems, which is the main focus of this paper, are caused by environmental factors such as supply voltage droops and temperature fluctuations. Voltage droops are caused in response to instantaneous current fluctuations due to activities on the power delivery network. Temperature fluctuation could alter the circuit parameters such as carrier mobility, threshold voltage, etc. Such variations can manifest themselves as timing errors, leading to incorrect computation results and system failures. Such variations have led to increasing use of overdesign and guardbands in circuit and architectural design to ensure reliability, which reduce the gains from process technology advances.

Due to the ability to adapt their learning parameters, neural networks have an inherent resilience to errors. Thus, one would expect that the quality of results produced by hardware neural networks (HNNs) to be relatively insensitive to the rising timing error rates caused by increased variation, thus opening doors for opportunistic reduction of guardbands to increase the operational efficiency of hardware. There is a need for a quantitative assessment here to explore the extent to which guardbands can be reduced in HNNs. In this paper, we investigate this question as to whether and how much accuracy of HNNs could be affected by dynamic variations. To do this, we capture and represent variations from low-level hardware, and then expose them to neural networks inferences. Unlike logic errors which can be derived though a mathematical formulation[8][22][28], variation-induced timing errors can only be obtained using gate-level simulation, making the error injection implementation time-consuming and not scalable.

Approach and Contributions: In this paper, we propose a cross-layer approach to assess the vulnerability of HNNs to dynamic voltage and temperature variations, in which we extract the timing errors from hardware layer using gate-level simulations and examine their effects in the software layer using error injections. To evaluate the soundness of this approach, we measure the timing errors using gate-level simulations (GLS) of a post-layout circuits in TSMC 45nm technology. We vary the voltage and temperature in a wide range to examine the effects of variations. Then, we represent and inject these timing errors to neural networks during their inference. Finally, we examine the resilience of two neural networks, MLP and CNN, by testing them on MNIST dataset[21].

Based on our implementation and evaluation, this paper makes the following contributions:

- We extract the circuit level timing errors caused by voltage and temperature variations from twenty different

978-1-5386-3093-8/17/$31.00 ©2017 IEEE

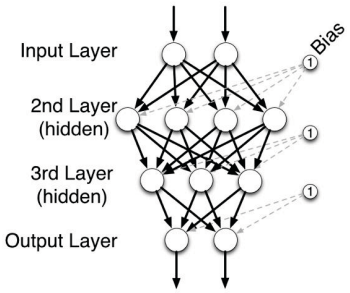

Fig. 1. An example of 4-layer multi-layer perceptron neural network.

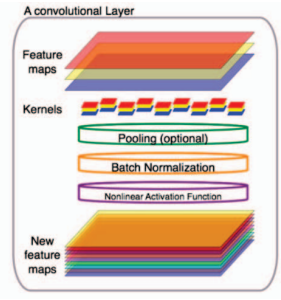

Fig. 2. The processes among a convolutional layer.

operating conditions using gate-level simulations.

- We inject such timing errors back into neural network inference and evaluate the accuracy on MNIST dataset at different conditions.
- Using two frequently used neural networks (MLP and CNN), our results show that variations can significantly affect the inference accuracy.

II. Hardware Neural Networks

Modeled for neural processing, Figure 1 shows a typical neural network, a multi-layer perceptron (MLP) consisting of an input layer, hidden layers, and an output layer. Except the input layer, all remaining layers are composed with artificial neurons that represent the basic computation unit. An artificial neuron consists of a linear processing part followed by a non-linear processing part. The linear part collects the output information, namely the activations, from previous layer. The collection method is a dot production between weights and activations. The nonlinear part includes regularization like dropout, and activation functions such as logistic sigmoid, hyperbolic tangent, or rectilinear unit. The nonlinear activation function enables a neural network to be a universal function approximator [10]. Rumelhart *et. al.* [23] intelligently applies the chain rule of calculus and gradient descent on neural networks to train the weights and hence minimizes the classification errors.

Since proposed in 1989, convolutional neural networks (CNNs) [20] have pushed the performance of neural networks to a new realm. Fig. 2 depicts the internal processes in a convolutional layer with 9 kernels, each of which consists of three filters. The convolution operation models the hardwired bonding between the neurons on adjacent layers. It uses a sliding filter to perform dot-products of the filter and uses a portion of the input image to generate an output image, namely the feature map. Since the convolution operations are differentiable, the filters can be trained to capture the features of the input images with backward propagation [23]. *Pooling* is used to reduce the size of a feature map by selecting the maximum pixel strength or averaging several pixel strengths. It benefits the transformation invariance because it drops unnecessary minor information and preserves the most dominant features for the overall classification task.

The robustness of a neural network comes from many aspects. From a higher level point of view, the training process of a neural network model is simply an ensemble of multiple linear or logistic regressions working in parallel. The regression itself ignores minor noises of the data and yields a model for the most likely distribution of the given data. The regularization process inside a neural network also contributes to robustness because no matter how deterministically penalties on weights are added or how stochastically certain partials of the model are dropped, the weights are trained to accommodate the majority of the data with a simplest probable distribution.

Hardware variations could impact HNNs through timing errors in both computation logic and control logic. The errors in control logic could lead to catastrophic results but fortunately, most critical paths lie in computation logic, which is mainly composed of additions and multiplications, two of the most frequently used operations. Both the forward and backward propagation require intensive additions and multiplications. Thus, in this paper, we mainly focus on the timing errors that occur in addition and multiplication.

III. Cross-layer Vulnerability Assessment

The cross-layer vulnerability assessment is comprised of two phases as shown in Fig. 3: *Timing Error Extraction and Timing Error Injection*. a) The *Timing Error Extraction* phase implements the standard ASIC flow and uses gate-level simulation (GLS) to generate timing errors at each operating condition. b) In the *Timing Error Injection* phase, we inject the timing errors into neural networks and then perform inference. We vary the neural network genres and operating conditions to examine the resulted accuracy. More details about the two phases are illustrated as follows.

A. HW-layer: Timing Error Extraction

We extract the timing errors through *Timing Error Extraction* module as illustrated in Fig. 3, which is divided into several steps. Note that we focus on dynamic variation-induced timing errors of computation units. We extract timing errors from the adder and the multiplier, which are the two most frequently used computation units in neural networks computation. We use FloPoCo [7] to generate the synthesizable VHDL codes of floating point units. We use *Synopsys Design Compiler* to synthesize the Verilog codes and use

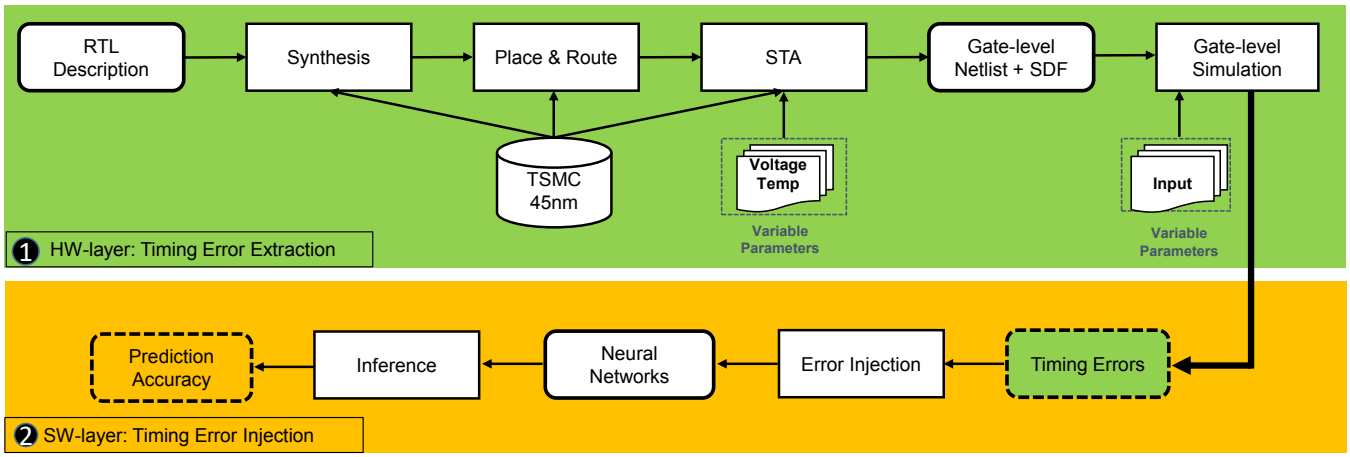

Fig. 3. Cross-layer assessment flow with two stages: a) HW-layer: Timing Error Extraction to extract the timing errors under different operating conditions; b) SW-layer: Timing Error injection into neural network and perform inference.

Synopsys IC Compiler to generate post place-and-route netlist in TSMC 45nm technology. Next, we use *Synopsys PrimeTime* to perform static timing analysis, generating Standard Delay Format (SDF) files at different operating conditions. To do this, we use the voltage temperature scaling features of Synopsys PrimeTime for the composite current source approach of modeling cell behavior. We consider twenty operating conditions as shown in Fig. 7, which could introduce both mild and aggressive timing errors. Then, we use *Mentor Graphics ModelSim* to do SDF back-annotation gate-level simulations under nominal frequency to generate output data at different operating conditions. To extract timing errors, we compare the GLS output $y[t]$ with a pure-RTL simulation result $y\_gold[t]$, which is free from timing errors because there is no delay annotation. If there is a mismatch, then we define it as a timing error.

B. SW-layer: Timing Error Injection

We inject the timing errors extracted by the *Timing Error Extraction* phase to the neural networks by using second phase *Timing Error Injection*. During the forward propagation in the neural network inference, we inject the errors into the computations (addition and multiplication). For a circuit, different input could excite different paths, resulting in an input-specific timing error behavior. To mimic this, an exhaustive look-up table containing the entire input space for each bit position of each computation unit under all operating conditions needs to be implemented. Then, the computations need to look up the table to check whether it has a match on any input operands in the input space. This makes the inference process prohibitively slow. To approximate the situation, we inject the timing errors as [24]: let the computation units return a random value each time they have timing errors. We inject the error into the computation with a TER extracted from the *Timing Error Extraction* phase to mimic the time error behavior. For example, if adder has a TER at 10%, we inject errors to 10% of the total additions. This probability is determined by operating conditions and computation logic (addition or multiplication), which can represent the impact of timing errors on computation logic. We vary the error injection probability for each operating condition.

IV. Experimental Results

In this section, we measure timing errors under twenty operating conditions. Then, we measure HNNs accuracy as a function of varying timing error rates. Finally, we characterize the HNNs accuracy under dynamic variations using MLP and CNN.

A. Experimental Setups

In this work, we use tiny-dnn [1], a header only, dependency free deep learning library written in C++, as our deep learning platform. This platform is light weighted, and is designed for deep learning on limited computational resource, such as embedded systems and IoT devices. For CNN, we use LeNet-5 like architecture and replace LeNet-5's RBF layer with normal fully-connected layer. For MLP, we use 3-layer MLP with a hidden layer of 60 neurons. We use MNIST (Mixed National Institute of Standards and Technology) database of handwritten numbers [21] as our dataset to evaluate the neural network accuracy. This dataset is a well-known dataset for evaluating the performance of neural network classifiers. The dataset is split into training set and test set with 60,000 and 10,000 28 $\times$ 28 images. We vary the voltage from 0.81V to 0.90V with a step at 0.01V and the temperature from 50°C to 100°C.

B. Accuracy under Timing Errors

We assess the accuracy for both MLP and CNN under the TER at 0, 0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, and 0.9 at three configurations as shown in Fig. 4 and Fig. 5; *add_only* means we only inject timing errors to adder, *mul_only* means

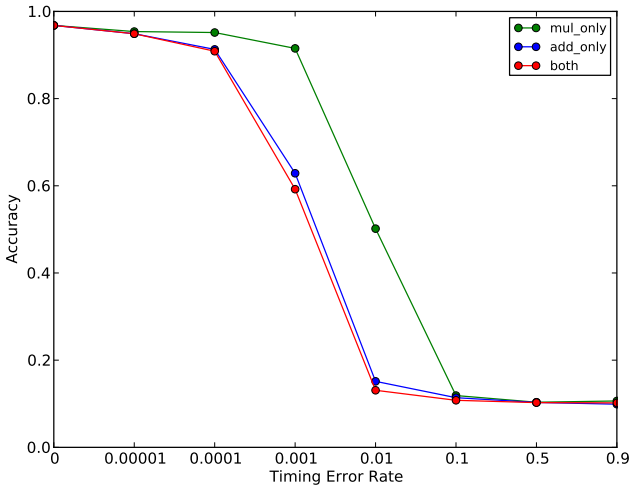

Fig. 4. MLP accuracy as a function of TER.

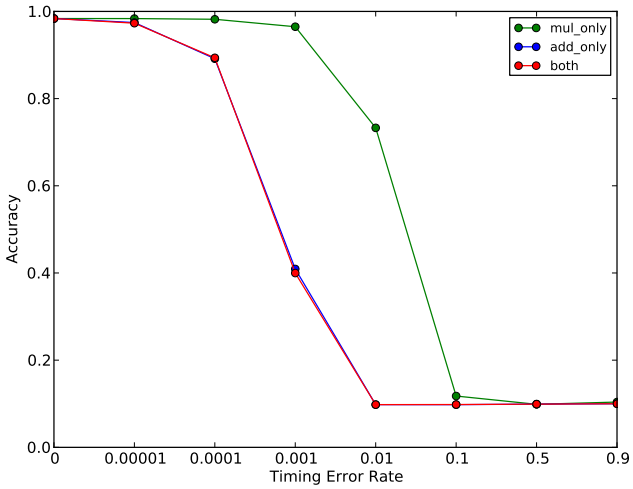

Fig. 5. CNN accuracy as a function of TER.

we only inject timing errors to multiplier and *both* means we inject errors to adder and multiplier at the same time. We observe that for both MLP and CNN, as the TER increases, the accuracy drops monotonically. When the TER is 0.00001, the HNN can still deliver a decent accuracy close to original accuracy. Once the TER of adder reaches 0.0001, the accuracy drops to around 90% and continue dropping to 60% when the TER of adder reaches 0.001. In contrast, the multiplier exhibits much less significant impact on HNN accuracy: the HNN can still deliver 90% accuracy even when the TER of multiplier reaches 0.001. In fact, for all examined TERs, the *mul_only* resulted accuracy is always higher than that of *add_only*. Moreover, the accuracy under *both* configuration is almost identical to that of *add_only* configuration, suggesting that adders-induced errors contribute to most of the accuracy drop. One main reason for this is that the accumulated convolution sum or dot-product sum are fed into a nonlinear activation function. The errors from multipliers will be averaged, but the errors from adders directly impact the input of the activation function. This suggests that more hardware design effort should be made on the adder to ensure its low TER. On the other hand, the worst accuracy of both NN genres is around 10%, when either *add_only* or *mul_only* is 0.1. We can observe that such an accuracy drop starts saturating at 0.1 TER, almost identical to a random guess, and stays almost the same when TER continues increasing. In summary, such observations show that even though neural networks have inherent error resilience, the timing errors still can significantly affect neural network accuracy and motivate this work.

C. Accuracy Versus Dynamic Variations

We then use the real dynamic operating conditions to obtain realistic timing error rates and thereby characterize the vulnerability of HNNs to dynamic variations.

First, we use the *Timing Error Extraction* described in Section III-A to characterize the timing error behavior of 32-bit floating point adder and multiplier under different operating

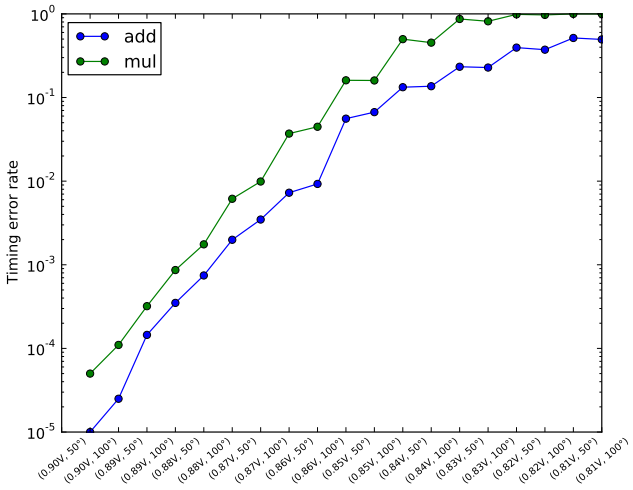

Fig. 6. TER of adder and multiplier under different operating conditions.

conditions as shown in Fig. 6. The selected operating conditions cover a wide range of TERs: at the best condition (0.90V, 50°C), no timing errors are injected for both computations; at the worst condition (0.81V, 50°C), 50% and 100% TER are found in adders and multipliers respectively. The TER of adder reaches 0.01 when the operating condition is around 0.86V. Based on Fig. 4 and Fig. 5, the accuracy drop starts to saturate when the TER of adder reaches 0.01, thus we expect to see a worst accuracy starting at around 0.86V.

We then present the accuracy of both MLP and CNN under twenty operating conditions, as shown in Fig. 7 and Table. I, where we observe several important facts. First, the lowest accuracy under worst-case operating conditions is around 10%. By looking into the prediction results, we found CNN is able to identify more than 90% of the *0* digits even under worst condition. Second, the 10% accuracy has been observed across multiple conditions from (0.86V, 50°C) to (0.81V, 100°C). (For better space utilization, we do not present the accuracy

TABLE I
HNN ACCURACY UNDER DYNAMIC VARIATIONS.

HNN	(0.90V, 50°C)	(0.90V, 100°C)	(0.89V, 50°C)	(0.89V, 100°C)	(0.88V, 50°C)	(0.88V, 100°C)
MLP	96.79%	96.03%	94.90%	87.93%	75.56%	57.76%
CNN	98.37%	97.31%	95.87%	85.15%	70.34%	48.64%
HNN	(0.87V, 50°C)	(0.87V, 100°C)	(0.86V, 50°C)	(0.86V, 100°C)	(0.85V, 50°C)	(0.86V, 100°C)
MLP	25.67%	15.89%	10.45%	10.33%	9.42%	9.91%
CNN	18.85%	11.13%	9.81%	9.80%	9.81%	9.81%
HNN	(0.85V, 50°C)	(0.85V, 100°C)	(0.84V, 50°C)	(0.84V, 100°C)	(0.83V, 50°C)	(0.83V, 100°C)
MLP	9.89%	9.80%	9.72%	9.60%	10.15%	9.60%
CNN	9.75%	9.81%	9.89%	9.80%	9.91%	9.84%

under 0.81V and 0.82V in Table. I, where the accuracy of both is around 10%.) This is expected as we can see from Fig. 4 and Fig. 5 where the accuracy drops to 10% when the TER of either unit reaches 0.1. Third, Table. I shows that under the condition between (0.86V, 50°C) and (0.90V, 100°C), where the TER of adder is less than 0.01, the accuracy drop of MLP to its original accuracy is less than that of CNN, indicating MLP might be more resilient than CNN within a certain TER. Part of the reason for this is that given the same TER, the amount of errors in CNN is larger than MLP because CNN has more arithmetic operations. Last but not least, we find the voltage and temperature both play an important role in determining the inference accuracy. By fixing the temperature at 100°C, reducing the voltage by 0.01V from 0.89V to 0.88V results an accuracy drop from 85.15% to 48.64%; by fixing the voltage as 0.88V, increasing the temperature by 50°C results an accuracy drop from 70.34% to 48.64%. By comparing the accuracy at (0.90V, 50°C) and (0.86V, 50°C), we find the accuracy drops to worst case at around 10% from best case at around 98% by a voltage reduction of 0.04V. In summary, such observations indicate there is a huge impact of dynamic variations on hardware neural networks accuracy and motivate the necessity of protecting HNN against variations.

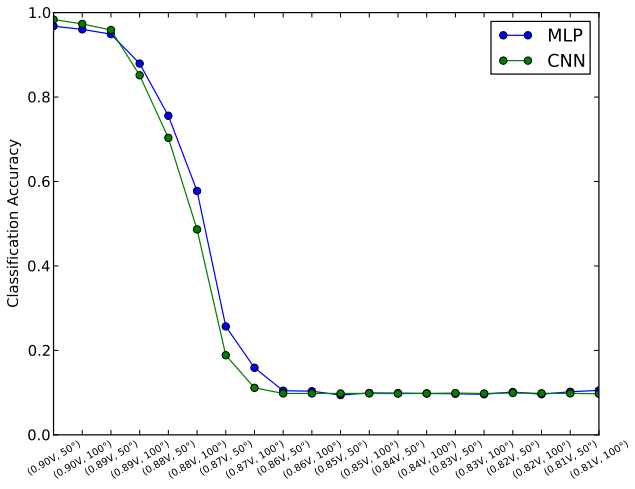

Fig. 7. HNN accuracy as a function of dynamic variations.

V. DISCUSSION

Threats to Validity: In this work, we mainly focus on variation-induced timing errors in computation logic. However, the timing errors could also occur in control logic, which might lead to more severe accuracy drop or malfunction. Fortunately, it was observed that control logic only contributes a small set of critical paths [25], making it less vulnerable to timing errors.
Future Work: In this work, we focus on assessing the effects of hardware variations on neural network performance. The next question is how we can mitigate such timing errors. For the future work, we focus on integrating the timing errors as a vector for backpropagation to enable an adaptive training method. Moreover, we plan to design a reconfigurable architecture that can automatically select suitable weights for a given voltage and temperature from a set of pre-stored weights.

VI. RELATED WORK

We describe the related work in three parts: combating timing errors, neural network resiliency and the main difference of our work with them.

Various hardware techniques have been developed to combat timing errors. Razor [9], uses a shadow flip-flop to detect timing errors and use recovery circuits to correct them. Error-detection sequential circuits (EDS) [2], double sample and compare signal arriving at different timing through such flip-flops and then correct them. Several learning methods are used to predict timing errors for functional units or instructions to enable an adaptive design [16][17][18]. A multi-armed bandit based optimization method was proposed to enable dynamic timing speculation [27]. Going up to the system level, bayesian networks have been used to calculate the system reliability with both hardware and software in consideration, and acquire higher accuracy [13][14].

More recent approaches to improving cost and energy efficiency have advocated tolerance to (and even use of) computational approximations, such as approximate adders [3][15]. These errors, originating from inexact logic design of computing unit, have been used in hardware neural networks to improve operational efficiency [8][22][28]. Du *et. al.* substitute the normal multipliers with inexact multipliers that provide inexact logic but with less hardware cost [8]. Mrazek *et. al.* further optimize such design with a uniform structure suitable for hardware implementation [22]. Xu *et. al.* provides

a framework for hardware neural network designers to choose which parts are suitable for approximation that leads to less impact on accuracy based on a criticality ranking [28]. These works intentionally design inexact hardware and introduce logic errors in exchange for less hardware cost.

Compared to logic errors, timing errors are less exploited in neural networks because of its unpredictability and uncertainty. Logic errors could be determined once the design is fixed but timing errors can only be obtained through simulations. A retraining-based method has been proposed to mitigate the timing errors in hardware neural networks [25]. However, these works assume a fixed timing variation for each gate without considering hardware variations as the root cause, which might be unrealistic.

In summary, there have been no prior works assessing the neural network vulnerability to dynamic variations. In this work, we do not introduce the errors intentionally but focus on the unintentional timing errors caused caused by hardware variations. We link the timing errors with low-level hardware variations and characterize them under different operating conditions and present the importance of considering variations when designing hardware neural networks.

VII. Conclusions

In this paper, we assess the effects of dynamic voltage and temperature variations on the performance of hardware neural networks. We first extract the timing errors of post place-and-route computation units under twenty operating conditions through gate-level simulations. We then inject such errors to neural network inference phase and evaluate the resulted accuracy. Using two frequently used neural networks, MLP and CNN, we demonstrate that dynamic voltage and temperature variations can cause significant drop in inference accuracy.

References

[1] tiny-dnn: header only, dependency-free deep learning framework in c++11.

[2] Keith Bowman et al. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *JSSC*, 2009.

[3] Vincent Camus, Jeremy Schlachter, and Christian Enz. A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision. In *Proceedings of the 53rd Annual Design Automation Conference*, page 127. ACM, 2016.

[4] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[5] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[6] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[7] Florent De Dinechin et al. Designing custom arithmetic data paths with flopoco. *IEEE Design & Test of Computers*, (4):18–27, 2011.

[8] Zidong Du, Avinash Lingamneni, Yunji Chen, Krishna V Palem, Olivier Temam, and Chengyong Wu. Leveraging the error resilience of neural networks for designing highly energy efficient accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1223–1235, 2015.

[9] Dan Ernst et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO-36.*, 2003.

[10] G Gybenko. Approximation by superposition of sigmoidal functions. pages 303–314, 1989.

[11] S Himavathi, D Anitha, and A Muthuramalingam. Feedforward neural network implementation in fpga using layer multiplexing for effective resource utilization. *IEEE Transactions on Neural Networks*, 18(3):880–888, 2007.

[12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[13] Yu Jiang et al. Bayesian-network-based reliability analysis of plc systems. *IEEE transactions on industrial electronics*, 2013.

[14] Yu Jiang et al. System reliability calculation based on the run-time analysis of ladder program. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, 2013.

[15] Xun Jiao, Vincent Camus, Mattia Cacciotti, Yu Jiang, Christian Enz, and Rajesh K Gupta. Combining structural and timing errors in overclocked inexact speculative adders. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 482–487. IEEE, 2017.

[16] Xun Jiao et al. Supervised learning based model for predicting variability-induced timing errors. In *Proc. of NEWCAS*. IEEE, 2015.

[17] Xun Jiao, Yu Jiang, Abbas Rahimi, and Rajesh K Gupta. Wild: A workload-based learning model to predict dynamic delay of functional units. In *Computer Design (ICCD), 2016 IEEE 34th International Conference on*, pages 185–192. IEEE, 2016.

[18] Xun Jiao, Yu Jiang, Abbas Rahimi, and Rajesh K Gupta. Slot: A supervised learning model to predict dynamic timing errors of functional units. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1183–1188. IEEE, 2017.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet Classification with Deep Convolutional Neural Networks. pages 1097–1105, 2012.

[20] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. 1.4:541–551, 1989.

[21] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[22] Vojtech Mrazek, Syed Shakib Sarwar, Lukas Sekanina, Zdenek Vasicek, and Kaushik Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *2016 International Conference On Computer Aided Design (ICCAD)(prijato)*, page 7, 2016.

[23] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. *CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE*, 1985.

[24] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, volume 46, pages 164–174. ACM, 2011.

[25] Ying Wang, Jiachao Deng, Yuntan Fang, Huawei Li, and Xiaowei Li. Resilience-aware frequency tuning for neural-network-based approximate computing chips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[26] Hongmei Yan, Yingtao Jiang, Jun Zheng, Chenglin Peng, and Qinghui Li. A multilayer perceptron-based medical decision support system for heart disease diagnosis. *Expert Systems with Applications*, 30(2):272–281, 2006.

[27] Jeff Jun Zhang and Siddharth Garg. Bandits: Dynamic timing speculation using multi-armed bandit based optimization. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 922–925. IEEE, 2017.

[28] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. Approxann: an approximate computing framework for artificial neural network. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 701–706, 2015.