Vulnerability of Hardware Neural Networks to Dynamic Operation Point Variations

Jeng-Hau Lin
Qualcomm, Inc.

Xun Jiao
Villanova University

Mulong Luo
Cornell University

Zhuowen Tu and Rajesh K. Gupta
University of California San Diego

Editor's notes:
This article studies the impacts of physical variations on neural networks. The proposed studies reveal an important observation that both multiple-layer perceptron (MLP) and convolutional neural network (CNN) may fail to operate appropriately even with small variations (e.g., voltage droops as small as 20 mV). Robust neural network architectures, including binarized neural network (BNN) and local binary pattern network (LBPNet), are explored to address this variability issue that has become a major bottleneck for practical applications.

—Xin Li, Duke University

Neural network (NN) algorithms have found use in a wide range of applications such as medical diagnostics, image classification, speech recognition, and natural language processing. This versatility has led to their implementation on a variety of hardware platforms: GPU, FPGA, and ASIC.

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 are the main focus of this article, 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 and threshold voltage. Such variations can manifest themselves as timing errors, leading to incorrect computation outputs and system failures. Notwithstanding setting up guardbands is the standard solution to ensure the system's functionality, the incomprehension of NNs' vulnerability can derive overdesigned guardbands encumbering the throughput of hardware accelerators or GPUs.

Due to the ability to adapt NNs' learnable parameters for extracting the abstract common features in data, NNs have an inherent resilience to errors. Thus, one would expect that the quality of results produced by hardware NNs (HNNs) to be relatively insensitive to the rising timing error rates (TERs) caused by increased variation, thereby opening doors for the 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. We investigate this question as to

Digital Object Identifier 10.1109/MDAT.2020.2986742
Date of publication: 9 April 2020; date of current version: 7 October 2020.

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 NN inferences. Unlike logic errors that can be derived through a mathematical formulation [2], variation-induced timing errors can only be obtained using gate-level simulation (GLS), making the error injection implementation time-consuming and not scalable.

Approach and contributions: 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 the hardware layer using GLSs and examine their effects on the software layer using error injections. To evaluate the soundness of this approach, we measure the timing errors using GLSs of postlayout circuits in TSMC 45-nm 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 NNs during their inference. Finally, we examine the resilience of four types of NNs: the multilayer perceptron (MLP), the convolutional NN (CNN), the binarized CNN (BCNN) [3], and the local binary pattern network (LBPNet) [5], [6], by testing them on the Modified National Institute of Standards and Technology (MNIST) data set.

Based on our implementation and evaluation, this article makes the following contributions.

- We extract the circuit-level timing errors caused by voltage and temperature variations from 20 different operating conditions using GLSs.
- We inject such timing errors back into the NN inference and evaluate the accuracy of the MNIST data set under different conditions.
- Our results quantitatively show that variations can significantly affect the inference accuracy on NNs.
- Among the four subject networks, LBPNet provides the more reliable error immunity than the other three networks.

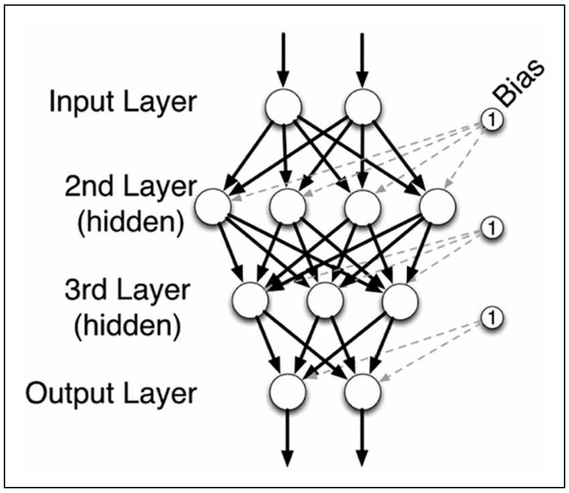

Figure 1. Example of a four-layer MLP NN.

Hardware neural networks

Modeled for neural processing, Figure 1 shows a typical NN, an MLP consisting of an input layer, hidden layers, and an output layer. Except for the input layer, all remaining layers are composed of artificial neurons that represent the basic computation unit. An artificial neuron consists of a linear processing part followed by a nonlinear processing part. The linear part collects the output information, also know as *activations*, from the previous layer, and 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, and rectilinear unit (ReLU). The nonlinear activation function enables an NN to be a universal function approximator. The forward–backward propagation algorithm intelligently applies the chain rule of calculus and gradient descent on NNs to train the weights and hence minimizes the classification errors.

Since proposed in 1989, CNNs have pushed the performance of NNs to a new realm. Figure 2 depicts the internal processes in a convolutional (Conv) layer with nine 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 the dot products of the filter and uses a portion of the input image gradually 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. Pooling is used to reduce the size of a feature map and increase the reception area by selecting the maximum pixel strength or averaging several pixel strengths. It benefits the translation invariance because it drops unnecessary minor information and preserves the most dominant features for the overall classification task.

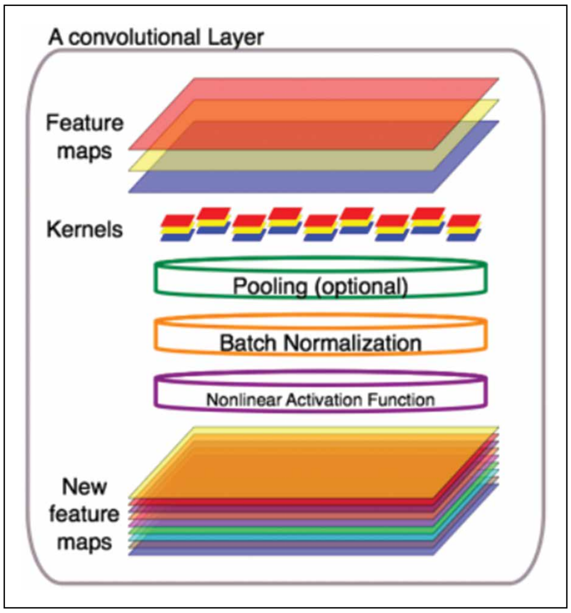

Figure 2. Processes among a Conv layer.

The robustness of NNs comes from many aspects. From a higher level point of view, the training process of an NN model is an ensemble of multiple linear or logistic regressions working in parallel. The regression ignores minor noises of the data and yields a model for the most likely distribution of the given data. Second, the regularization process inside an NN also contributes to robustness because weights are deterministically penalized if the tensor norms grow too large and the connections can be dropped stochastically to elude a network learning unwanted noises. The weights are, thereby, trained to accommodate the majority of the data with the simplest probable distribution. Moreover, if a re-training process is involved, the convex optimization enforces the learnable parameters in a model to descend on the error surface again. Please note that we only assess the inference performance in this work without performing any re-training.

Binarized NN (BNN) [3] was proposed as an extreme case of network quantization. During the training phase, it maintained two sets of weights: The one set of weights contained floating-point numbers to guarantee a smooth gradient descent, and the other set was the binarized one obtained by a hard-hyperbolic tangent function that returned “+1” if the input was positive; otherwise, returned “–1.” The forward propagation used the binarized weights to predict network output and calculate loss, and the backward propagation relied on the floating weights to descend the model on a smooth error surface. Whenever the floating-numbered weights got updated, they were binarized and stored in the binarized weights. However, given that the binarized weights cannot carry sufficient information for most classification tasks, a small number of floating number calculations were introduced to compensate for the information loss, i.e., both bias addition and batch normalization were in floating numbers.

An LBPNet [5], [6], as shown in Figure 3, was proposed as an alternative deep learning method to CNN for optical character recognition tasks. Instead of using multiplication-and-accumulation (MAC) operations, local binary pattern (LBP) operation [9] leveraged sampling and comparison to efficiently capture features. Gupta et al. [5] and [6] further introduced LBP to deep learning by stacking the LBP layers together, applying random projection to avoid channel accumulation, and deriving the calculus chain rule to develop LBPNet’s backward propagation. For optical character recognition tasks, local binary pattern (LBP) Nets delivered near state-of-the-art classification accuracy while reducing the computation demand and model size of Conv layers by two to three orders of magnitudes. In this work, we also binarized the fully connected (FC) layers of an LBPNet for the test of vulnerability.

Hardware variations could impact HNNs through timing errors in both computation logic and control logic. The errors in control logic could lead to

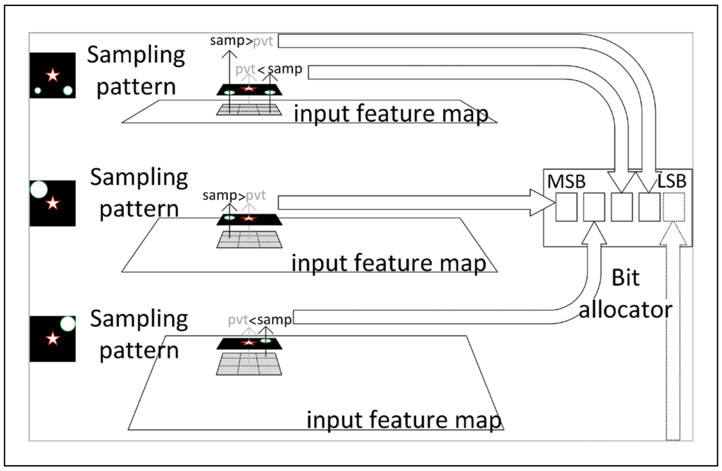

Figure 3. Detailed illustration of an LBP layer. Three LBP patterns work like masks for sampling through the pivot aperture (pvt) and sampling apertures (samp). The comparison results are allocated to a bit array according to the random projection map.

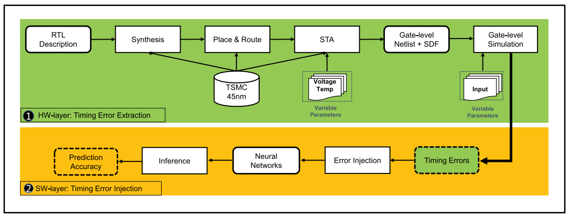

Figure 4. Cross-layer assessment flow with two stages. (a) Hardware layer: timing error extraction to extract the timing errors under different operating conditions. (b) Software layer: timing error injection into an NN and perform inference.

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 propagations require intensive additions and multiplications, but most HNNs on ASIC, FPGA, and embedded GPUs do not support on-chip learning. Thus, we mainly focus on the timing errors that occur in addition and multiplication during the inference phase of HNNs.

Cross-layer vulnerability assessment

The cross-layer vulnerability assessment is comprised two phases, as shown in Figure 4: 1) timing error extraction and 2) timing error injection. The timing error extraction phase implements the standard ASIC flow and uses GLSs to generate timing errors under each operating condition. In the timing error injection phase, we inject the timing errors into NNs and then perform inference. We vary the NN genres and operating conditions to examine the resulted accuracy. More details about the two phases are illustrated as follows.

Hardware layer: timing error extraction

We extract the timing errors through the timing error extraction module, as illustrated in Figure 4, 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 NN computation. We use floating-point cores [1] to generate the synthesizable VHDL codes of floating-point units. We use the *synopsys design compiler* to synthesize the Verilog codes and use the *synopsys IC compiler* to generate the post-place-and-route netlist in TSMC 45 nm technology. Next, we use Synopsys PrimeTime to perform static timing analysis, generating standard delay format (SDF) files under 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 20 operating conditions, as shown in Figure 8, which could introduce both mild and aggressive timing errors. Then, we use Mentor Graphics ModelSim to do SDF back-annotation GLSs under nominal frequency to generate output data under 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.

Software layer: timing error injection

We inject the timing errors extracted by the timing error extraction phase to the NNs by using the second phase timing error injection. During the forward propagation in the NN inference, we inject the errors into the arithmetic computations (addition and multiplication) in the Conv layer, FC layer, average pooling layer, batch normalization (BatchNrom) layer and LBP layer. There are several noteworthy facts that must be highlighted regarding the error injection in the software layer. First, the XNOR operation and pop-count accumulation in BCNN and the comparison operation in an LBP layer are not implemented in conventional arithmetic and logic units on CPUs or processing elements on GPUs. We have to use multipliers and adders to carry out the 1-bit XNOR and the following accumulation in BCNN. For the comparison in an LBP layer, we use the sign bit of subtraction to produce the comparison result instead. Therefore, the TER from adders and multipliers can affect the outputs of binarized Conv, binarized FC, and LBP layers.

On 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 [10]: let both the *mul_only* and *add_only* computation units return a random value each time they have timing errors. We inject the error into the computation with the pair of adder TER and multiplier TER extracted from the timing error extraction phase to mimic the time error behavior. For example, if the adder has a TER at 0.1, 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.

Experiments

In this section, we measure timing errors under 20 operating conditions. Then, we measure the HNN accuracy as a function of varying TERs. Finally, we characterize the HNN accuracy under dynamic variations using MLP, CNN, BCNN, and LBPNet.

Experimental setups

In this work, we use tiny-dnn [8], a header-only, dependency-free deep learning library written in C++, as our deep learning platform for MLP and CNN. This platform is light weighted and is designed for deep learning on the limited computational resource, such as embedded systems and Internet of Things devices. For CNN, we use the LeNet-5 like architecture and replace the LeNet-5's RBF layer with an FC layer. For MLP, we use a three-layer MLP with a hidden layer of 60 neurons. We adopt the same structure of BCNN for Street View House Numbers in the BNN paper and the LBPNet for MNIST in the LBPNet paper [5]. The synthesizable C codes for BCNN and LBPNet implemented by us for FPGA accelerators are used for the error injection. All the four sets of weights and kernels are pretrained either from the referred tiny-dnn source or by us on an Nvidia Tesla K40 GPU.

We use MNIST and CIFAR-10 as our data sets to evaluate the NN accuracy. MNIST of handwritten numbers is a well-known data set for evaluating the performance of NN classifiers. The MNIST data set has a training set and a test set of 60,000 and 10,000 with each having 28×28 pixel images. The main features of the MNIST data set are strokes and outlines. Images in CIFAR-10 are daily objects of size 32×32, and the training and test sets include 50,000 and 10,000, respectively. CIFAR-10 is considered to be a more challenging data set because its information and features reside in both outlines and textures. We choose MNIST to conduct a decent evaluation of vulnerability as the first step. Then, we deepen and widen BCNN and LBPNet by adding more layers, kernels, and random projection maps to conduct the second step of experiment on CIFAR-10 to understand the vulnerability of HNN on general object recognition.

For hardware variations, we vary the voltage from 0.81 to 0.90 V with a step at 0.01 V and the temperature from 50 °C to 100 °C.

Accuracy under the threat of timing errors

Before the error extraction, we assess the performance degeneration as a function of TERs. The accuracy is evaluated 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 Figures 5 and 6; *add_only* means that we only inject timing errors to adder, *mul_only* means that we only inject timing errors to multiplier, and both means that we inject errors to the adder and the multiplier at the same time. We observe that for both MLP and CNN, as the TER increases, the accuracy drops monotonically. When the TER reaches 0.00001, the HNN can still deliver a decent accuracy close to original accuracy. Once the TER of the adder reaches 0.0001, the accuracy drops to around 90% and continues dropping to 60% until the TER of the adder the reaches 0.001. In contrast, the multiplier exhibits a much less significant impact on the HNN accuracy: the HNN can still deliver 90% accuracy even when the TER of the multiplier reaches 0.001. In fact, for all examined TERs, the resultant accuracy of *mul_only* 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 behind the accuracy drop is that the accumulated convolution sum or the dot-product sum is fed into a nonlinear activation function, thereby directly affecting the activation, whereas the errors from multipliers are averaged and diluted. 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* reaches 0.1. We can observe that such an accuracy drop starts saturating at 0.1 TER, almost identical to a random guess of the ten-class recognition task.

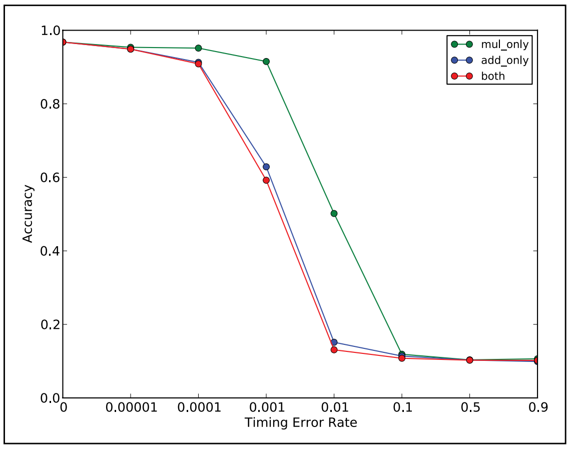

Figure 5. MLP accuracy as a function of TER.

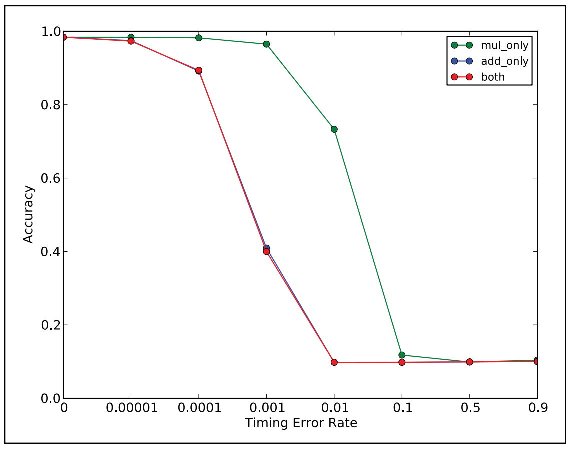

Figure 6. CNN accuracy as a function of TER.

Another important observation is that the accuracy of CNN decreases more drastically than that of MLP, which conflicts our intuition of the higher capability of CNN. The classification accuracy at the adder-only 0.001 TER is 61%, which is higher than CNN's 40% accuracy at the 0.001 TER. However, when we inspect in detail, the fan-in of a neuron and a convolutional kernel reveals the surprising observation. The fan-in of a convolutional kernel is defined by the spatial size of a filter, which is 3×3 and relatively small compared to a neuron's fan-in of MLP. Therefore, the injected error in MLP gets diluted better.

In summary, such observations show that even though NNs have inherent error resilience, the timing errors can still significantly affect the NN accuracy and motivate this work.

Vulnerability of the MNIST

We use the real dynamic operating conditions to obtain realistic TERs, thereby characterizing the vulnerability of HNNs to dynamic variations. Notably, we use the timing error extraction phase described in the "Hardware layer: timing error extraction" section to characterize the timing error behavior of a 32-bit floating-point adder and a multiplier under different operating conditions, as shown in Figure 7. Besides the ideal condition without any error, the selected operating conditions cover a wide range of TERs: at the best condition (0.90 V and 50 °C) with TERs less than 0.0001; under the worst condition (0.81 V and 50 °C), the 0.5 and 1.0 TERs are found in adders and multipliers, respectively. By comparing these two computing units, we find that the TER of the multiplier is always higher than that of the adder under the same condition. This is because the multiplier design has more critical paths than the adder, resulting in more timing violations. The TER of the adder reaches 1% when the operating condition is around 0.86 V. As shown in Figures 5 and 6, the accuracy drop starts to saturate when the TER of the adder reaches 0.01; thus we expect to see the worst accuracy starting at around 0.86 V.

We then present the accuracy of MLP, CNN, BCNN, and LBPNet under the 20 operating conditions, as shown in Figure 8, where we observe several important facts as follows.

First, the lowest accuracy under the worst-case operating conditions is around 10% for all the four networks across multiple conditions from (0.85 V and 100 °C) to (0.81 V and 100 °C). For MLP, CNN, and BCNN, this observation is expected as we can see from Figures 5 and 6 where the accuracy drops to 10% when the TER of either unit reaches 0.01.

Second, the four curves can be categorized into two groups because MLP, CNN, and BCNN behave similarly, and the LBPNet's accuracy curve demonstrates a high immunity to the TER residing in adders and multipliers.

Third, Figure 8 shows that under the condition between (0.90 V and 50 °C) and (0.86 V and 50 °C), where the TER of the adder is less than 0.01, the accuracy drop of MLP to its original accuracy is less than that of CNN, indicating that 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, and the percentile of multiplications among all arithmetic computations is higher in CNN.

Fourth, BCNN sustains slightly more timing errors than MLP and CNN. Compared with MLP's curve, BNN's vulnerability is enhanced twice since the classification drops to the same with MLP when the TER is doubled.

Fifth, LBPNet keeps immune against the variation until we impose much harsher conditions. A 10% accuracy deterioration is observed at 0.86 V and 50 °C, whereas all the other three models significantly lose classification ability and fall around 10% accuracy. LBPNet totally fails to classify upon 0.85 V and 100 °C, as the TERs climb to 0.1 and 0.5 for adders and multipliers, respectively.

Sixth, last but not least, we find that both the voltage and temperature play an important role in determining the inference accuracy. Fixing the temperature at 100 °C and reducing the voltage by 0.01 V from 0.89 to 0.88 V results in an accuracy drop of the CNN model from 85.15% to 48.64%; fixing the voltage at 0.88 V and increasing the temperature by 50 °C results in an accuracy drop from 70.34% to 48.64%. By comparing the accuracy at 0.90 V and 50 °C and 0.86 V and 50 °C, we find the accuracy drops to the worst case at around 10% from the best case at around 98% by a voltage reduction of 0.04 V.

Vulnerability of CIFAR-10

Figure 9 shows the result of CIFAR-10. In the second step, besides deepening and widening BCNN and LBPNet, we reduced the size of MLP classifiers to two layers of 512 and 10 neurons. Only one BatchNrom layer is preserved so that the training process is accelerated and the vulnerability of the binarized Conv layers and LBP layers can be more prominent. The structure of BCNN is the same as the structure of the original BNN paper except for the simplified FC classifier. We stack the LBPNet to 10 layers and utilized an ensemble of 15 sets of random projection maps to achieve a competent accuracy with BCNN.

The initial accuracy of BCNN and LBPNet is around 81%. As the hardware variation increases,

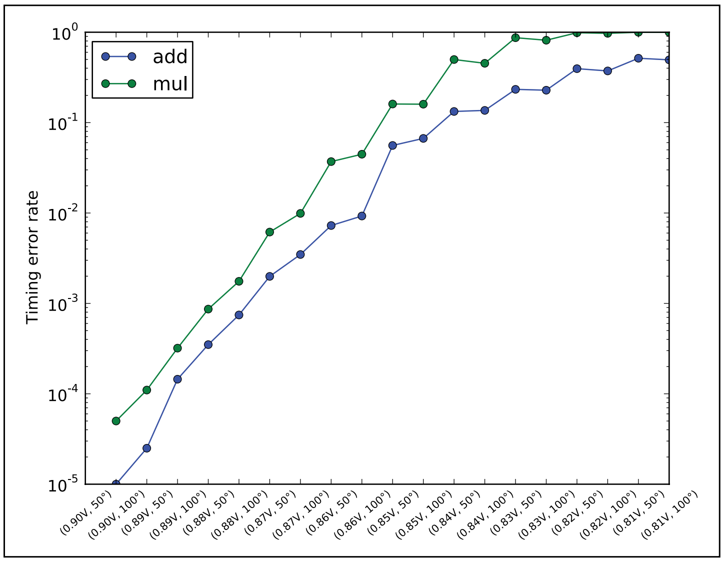

Figure 7. TER of adder and multiplier under different operating conditions.

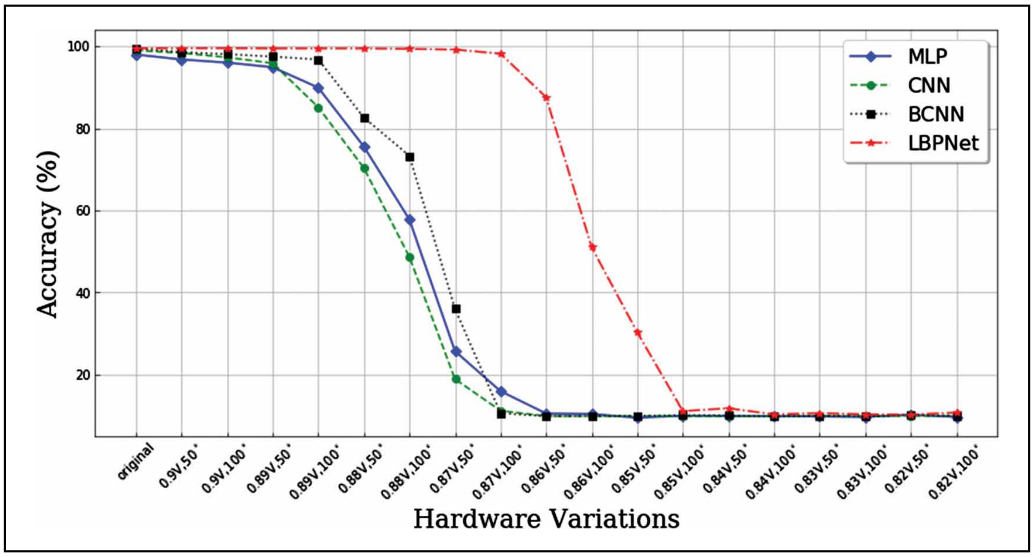

Figure 8. HNN accuracy as a function of dynamic variations on MNIST.

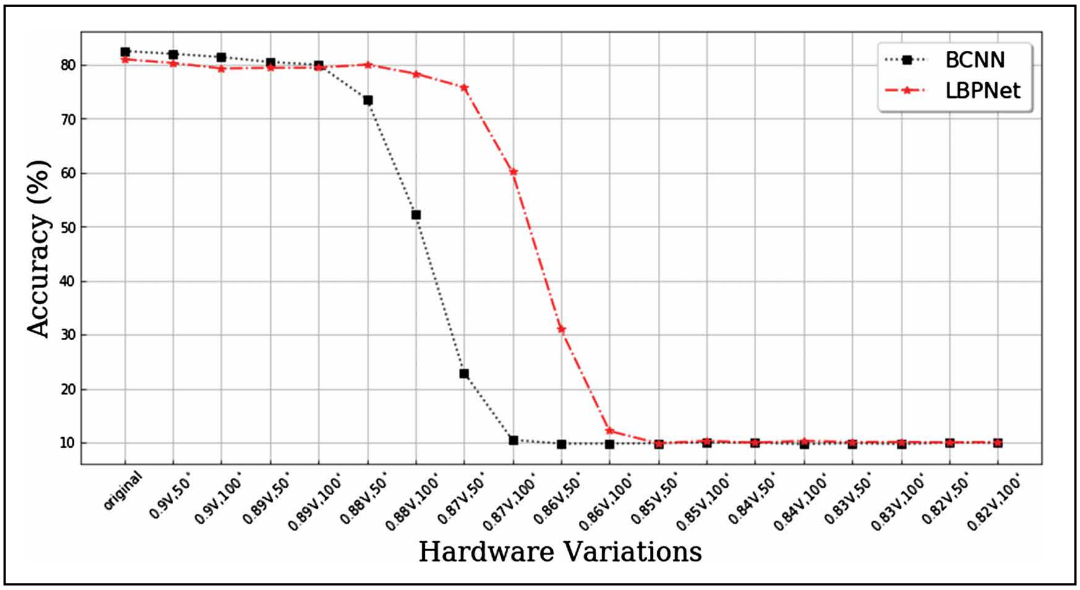

Figure 9. HNN accuracy as a function of dynamic variations on CIFAR-10.

BCNN's classification ability start to degrade after 0.89 V and 50 °C, which is not far from the result of the first experiment. However, the immunity of the deeper and wider LBPNet becomes less robust since LBPNet's accuracy starts to fall off of the cliff at 0.88 V and 50 °C. In other words, if Figures 8 and 9 are overlapped, we can see that the curves of BCNN and LBPNet recede into the left, and the extent of degradation for LBPNet is more obvious. There is, however, still a gap between the two curves. Although the gap is reduced, the depth of BCNN remains the same. The classifier in a deeper network would collect more errors than that in a shallow network.

Discussion on the BNN and the LBPNet

The binarized values and operations in BCNN rectify a portion of the injected errors, thereby enhancing the robustness. Specifically, the 1-bit multiplication and 1-bit accumulation again dilute the impact of the injected errors. Moreover, when the binary activation function converts the erroneous inner product sum or convolution sum, only the sign inversion changes the activation output. That is, the total of injected errors collected by the activation function must be strong enough to invert the sign; otherwise, the activation output remains the same without the error injection.

The immunity of LBPNet outperforms the other models with a remarkable gap. There are multiple causes that contribute to this immunity, which can be qualitatively justified by revisiting the details of an LBP Layer. The comparison is simulated with the sign bit from the adder's subtraction output. Then, the sign bits corresponding to an LBP kernel are produced by adders in parallel and form a bit sequence to represent an integer on the output feature map. Whenever the adder is stochastically selected for an error injection, the sign bit is flipped randomly according to a uniform distribution. Therefore, an injected error can only affect a single bit rather than an output value as in MLP and CNN. Furthermore, if the selected bit is not the most significant bit of the output value, the effect of error injection is negligible. On the other hand, the sign bits are combined with a bit shift and a logic OR operations in parallel, which are relatively less affected by the hardware variations given their circuitry simplicity and are not within the scope of this work. The absence of accumulation helps LBPNet to preclude the error accumulation and hinders the propagation of errors.

Related work

Various works study the vulnerability of HNNs under logic errors induced by inexact design [2], [7], [12]. Du et al. [2] substituted the regular multipliers with inexact multipliers that provide the inexact logic but with less hardware cost. Mrazek et al. [7] further optimized such design with a uniform structure suitable for hardware implementation.

Zhang et al. [12] provided a framework for hardware NN designers to choose which parts were suitable for an approximation that led to less impact on accuracy based on a criticality ranking. These works intentionally designed inexact hardware and introduced logic errors in exchange for less hardware cost.

Compared to logic errors, timing errors were less exploited in neural networks because of its unpredictability and uncertainty [4]. 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 [11]. However, these works assumed 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 NN vulnerability to dynamic operating condition variations. In this work, we do not introduce the errors intentionally but focus on the unintentional timing errors 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 NNs.

Threats to validity: 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 [11], making it less vulnerable to timing errors.

Promising Solutions: The observations and discussions in the previous sections have enlightened us about several directions to strengthen the immunity of the voltage and temperature variations.

- Considering the experiments on MLP and CNN, we can increase the fan-in of each accumulation to dilute the impact of hardware variations. However, this trend conflicts with pruning, which is a prevailing model reduction method. People need to be aware of the fact that the side effects of pruning include the degradation of network vulnerability.
- Binarizing the network also dilutes the timing error's impact, and it works for both MLP and BCNN. More generally, the quantization of a network not only makes the network hardware-friendly but also increases its vulnerability.
- Although deepening a network structure can usually increase classification accuracy, we need to keep in mind that the increase of depth will reduce the immunity to hardware variations.
- Another method is to adopt LBPNet because the lack of floating number MAC operations and the high parallelism in the LBP operations have demonstrated that both the error injection and propagation in LBPNets can be limited effectively.

Future work: In this work, we focus on assessing the effects of hardware variations on NN performance. The next question is: How can we mitigate such timing errors? For future work, we focus on integrating the timing errors as a vector for backward propagation 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 prestored weights. ■

■ References

[1] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design Test. Comput.*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

[2] Z. Du et al., "Leveraging the error resilience of neural networks for designing highly energy efficient accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 8, pp. 1223–1235, Aug. 2015.

[3] I. Hubara et al., "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[4] X. Jiao et al., "SLoT: A supervised learning model to predict dynamic timing errors of functional units," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1183–1188.

[5] J.-H. Lin et al., "Local binary pattern networks," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2020, pp. 825–834.

[6] J.-H. Lin et al., "Accelerating local binary pattern networks with software-programmable FPGAs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 1112–1117.

[7] V. Mrazek et al., "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proc. 35th Int. Conf. Comput.-Aided Design (ICCAD)*, 2016, p. 7.

[8] T. Nomi. (2016). *Tiny-DNN*. Accessed: Apr. 10, 2017. [Online]. Available: https://github.com/nyanp/tiny-cnn

[9] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern Recognit.*, vol. 29, no. 1, pp. 51–59, Jan. 1996.

[10] A. Sampson et al., "EnerJ: Approximate data types for safe and general low-power computation," *ACM SIGPLAN Notices*, vol. 46, no. 6, p. 164, Jun. 2011.

[11] Y. Wang et al., "Resilience-aware frequency tuning for neural-network-based approximate computing chips," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2736–2748, Oct. 2017.

[12] Q. Zhang et al., "ApproxANN: An approximate computing framework for artificial neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 701–706.

Jeng-Hau Lin is a Research Engineer at Qualcomm Inc. He nurtures a keen interest in computer architecture on mobile devices for the deep learning algorithm.

Xun Jiao is an Assistant Professor with Villanova University, Villanova, PA. His research interests include edge computing, machine learning, and embedded systems. He is an Associate Editor for the IEEE Transactions on Computer-Aided Design.

Mulong Luo is currently pursuing a PhD with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY. His research interests include computer architecture and cyber–physical system security.

Zhuowen Tu is a Professor of Cognitive Science, University of California, San Diego, La Jolla, CA. His main research interests include computer vision, machine learning, and neural computation. Tu has a PhD in computer science from Ohio State University, Columbus, OH. He is a Fellow of the IEEE.

Rajesh K. Gupta is a Professor in the Department of Computer Science and Engineering and the Founding Director of Halıcıoğlu Data Science Institute at the University of California, San Diego, La Jolla, CA. He leads the Microelectronic Embedded Systems Lab, whose research interests span topics that have recently come to be characterized under embedded, cyber–physical systems and more recently under Internet-of-Things. He is a Fellow of IEEE, ACM, and AAAS.

■ Direct questions and comments about this article to Jeng-Hau Lin, Computer Science and Engineering Department, University of California San Diego, La Jolla, CA 92093-0021 USA; jel252@ucsd.edu.