Resilient neural network training for accelerators with computing errors

Dawen Xu*†, Kouzi Xing†, Cheng Liu*[1], Ying Wang*, Yulin Dai†, Long Cheng‡, Huawei Li* and Lei Zhang*
*Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
†Hefei University of Technology, Hefei, China
‡University College Dublin, Dublin, Ireland
Email: xudawen@gmail.com {liucheng, wangying2009, lihuawei, zlei}@ict.ac.cn long.cheng@ucd.ie

***Abstract*—With the advancements of neural networks, customized accelerators are increasingly adopted in massive AI applications. To gain higher energy efficiency or performance, many hardware design optimizations such as near-threshold logic or overclocking can be utilized. In these cases, computing errors may happen and the computing errors are difficult to be captured by conventional training on general purposed processors (GPPs). Applying the offline trained neural network models to the accelerators with errors directly may lead to considerable prediction accuracy loss.**

To address this problem, we explore the resilience of neural network models and relax the accelerator design constraints to enable aggressive design options. First of all, we propose to train the neural network models using the accelerators' forward computing results such that the models can learn both the data and the computing errors. In addition, we observe that some of the neural network layers are more sensitive to the computing errors. With this observation, we schedule the most sensitive layer to the attached GPP to reduce the negative influence of the computing errors. According to the experiments, the neural network models obtained from the proposed training outperform the original models significantly when the CNN accelerators are affected by computing errors.

I. Introduction

Inspired by the widespread adoption of neural networks in massive fields, neural network accelerators [1], [2] are increasingly explored and deployed to improve the computing performance and energy efficiency. Unlike generic applications, neural networks usually involve redundancy and are known to be fault tolerant [3]. By taking advantage of this feature, many neural network accelerator optimizations such as neural network pruning and quantization can be utilized to improve performance and energy efficiency notably with minor inference accuracy penalty [4].

Following similar ideas to compromise between neural network accuracy and performance, we opt to relax the design constraints of the neural network accelerators for significant performance or energy efficiency improvements with minor prediction accuracy loss. In this case, many aggressive hardware optimization techniques can be applied when computing errors can be tolerated. For instance, emerging techniques such as near-threshold voltage regime and sub-threshold digital logic design [5] promise high energy efficiency but suffer instability [6]. Conventional neural network accelerator can be pushed to operate at higher clock frequency with timing violations [7]. They will bring benefits to the accelerator design on various aspects including performance and energy efficiency.

Motivated by the great advantages of relaxed design constraints, we further explore the use of neural network resilience for more effective design trade-offs. Instead of deploying the unmodified neural network models on the accelerators directly, we borrow the retraining idea from prior neural network quantization work [8] and have the deep neural network models to learn and tolerate the computing errors. Basically, we have the forward computing performed on the accelerator and then transfer the computing results to the host processor for backward propagation. With this approach, both the application data and computing errors are learned and incorporated in the neural network models. Meanwhile, we define a set of standard interfaces to make it convenient to integrate general CNN accelerators into the retraining framework.

In addition, we notice that some of the neural network layers are more sensitive to the computing errors and the sensitive layers dramatically limit the usefulness of the retraining. Thus, we schedule the most sensitive layer of the neural networks to host processors to reduce the negative influence of the computing errors. With both the retraining and sensitive layer protection, the neural networks become more resilient to the computing errors caused by the aggressive design options such as near-threshold logic or overclocking. Compared to the original neural networks, the top1 and top5 prediction accuracy improves by 20.7% and 5.9% on average.

II. Resilient training framework

Resilient neural networks allow significant performance or energy efficiency improvement with little prediction accuracy penalty by relaxing the neural network accelerator design constraints. This motivates us to obtain more resilient neural networks for advantageous accelerator design trade-offs. A resilient neural network training framework will be detailed in this section.

A. Overall training framework

For the problem that the computing error patterns are difficult to be captured in training with GPPs, we have the accelerators with computing errors integrated into the training

[1]Corresponding author is Cheng Liu.

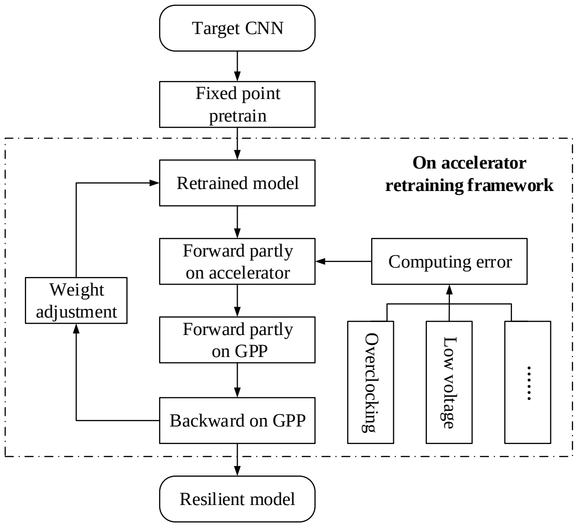


Fig. 1. Resilient neural network training framework

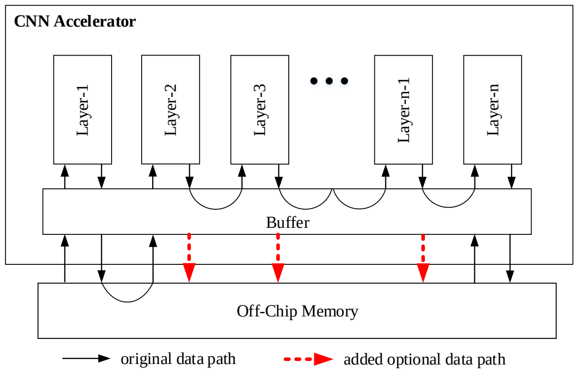


Fig. 2. Modification of the CNN accelerator data path. It essentially ensures the feature map of each neural network layer to have an optional data path to external memory for back propagation in training.

process. Forward processing influenced by the accelerator computing errors is used in training directly such that computing error patterns and application data are reflected in the neural network models. For the problem that some of the layers are affected more than the others, we take these layers as critical layers and opt to protect the layers from being affected by computing errors. With reasonable performance penalty, we can improve the overall neural network resilience.

Following this idea, the overall training framework is depicted in Figure 1. Instead of training on GPPs, it has the majority of forward computing performed on the accelerators with computing errors while the rest of the training remains on GPPs. Note that the critical layers should be executed on reliable hardware while GPP is one of the options. There are many different approaches that can be used to relax the design constraints. Although they may cause distinct computing errors, they can be fitted to the same training framework.

B. CNN accelerator abstraction and modification

As illustrated in the above section, the forward propagation will mostly be executed on the CNN accelerator while the rest part runs on GPPs. Essentially, the framework targets at a heterogeneous computing architecture and frequent communication between the accelerator and the GPPs are expected. In order to fit various CNN accelerators within the same training framework, we abstract the CNN accelerators with a high-level interface which makes the accelerators near transparent to the training framework. To facilitate the data communication between the forward propagation and the rest of the training framework, we define a high-level interface which consists of 7 functions as listed inTable I. With the interface functions, general CNN accelerators can be conveniently referenced and used in the proposed on-accelerator training framework.

In this work, we have the CNN accelerator implemented on Xilinx FPGAs as a case study. With Xilinx SDAccel, we can wrap the accelerators with OpenCL API while the accelerators can either be developed with OpenCL, HLS or RTL. On top of the OpenCL API, the proposed high-level interface can be implemented. Meanwhile, we use Caffe, a C++ based deep learning framework, to construct the on-accelerator training framework. With both parts developed with C family languages, they can be integrated conveniently.

On top of the high-level interface, the CNN accelerator also needs minor modification to enable the on-accelerator training. The training requires the feature map of each neural network layer for backward propagation. However, many of the accelerators are intensively optimized for inference only and some of the layers' output are fully buffered in on-chip memory to reduce the external memory access. Thereby, the accelerator should provide an optional data path such that intermediate output data can be written to external memory at request. As shown in Figure 2, the output of each layer will be transferred to memory using the added data path when the accelerator is used for training. The write back data path can be switched off during inference.

C. Critical neural network layer protection

In order to improve the overall neural network resilience, we opt to protect the critical network layers to alleviate the resilience bottleneck. The protection is essentially to have the critical layers executed on reliable computing infrastructures and the exact protection method depends on the target hardware platform. We may either schedule the critical layers to the GPPs or switch the accelerator to reliable mode during the execution of the critical layers. With this approach, the overall network can tolerate more computing errors.

To decide the critical layers of the neural networks, we formulate the critical layer selection scheme. Suppose the neural network layers include N layers and each layer is represented as L_i where $i \in 0, 1, 2, ..., N-1$. Then we evaluate the prediction accuracy loss of the neural network that have one layer protected on accelerators with computing errors. When the ith layer is protected, the loss is $loss_i$. Then the most critical layer is the layer that leads to the most accuracy loss i.e. $c \in \{k | loss_k = max(loss_i), i \in \{0, 1, ..., N-1\}\}$.

The above formulated approach requires large amount of evaluation of different layers of the neural network. Instead, we

TABLE I
HIGH-LEVEL INTERFACE TO INTEGRATE GENERAL CNN ACCELERATORS WITH CAFFE

ID	Function Name	Description
1	launchAccelerator()	It configures the CNN accelerator and launches it from host CPU.
2	dataToFPGA(weight, input, wgtDevAddr, inDevAddr)	It transfers both the input data and weight to the FPGA device memory.
3	dataFromFPGA(outputDevAddr, output)	It transfers intermediate data from FPGA device memory to host memory.
4	convertIntToFloat(int iData, float fData)	It converts the fixed-point point to float for back propagation processing.
5	convertFloatToInt(float fData, int iData)	It converts the floating-point input and weight to fixed point for forward processing.
6	dataLayoutReorder(data, reorderedData)	It reorders the data layout for more efficient accelerator execution.
7	dataLayoutRecover(reorderedData, data)	It reorders the output data back to the default format for Caffe back propagation.

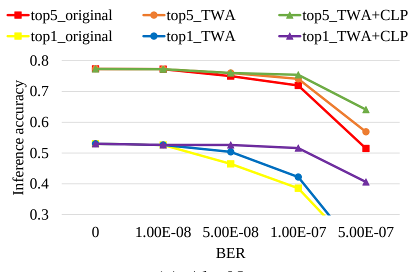
use the actual computing errors as the critical layer selection metric. We set an error threshold T and assume 8bit integers are used. When the error equals to 0, the computing results are correct. When the error is larger than T, the results are assumed to be large errors. When the computing results are wrong but smaller than T, the results are considered as moderate errors. The layers that include the largest portion of large errors are taken as the critical layers.

In addition, scheduling the neural network layers executed on the accelerator to GPPs has performance penalty due to the computing efficiency gap. As the accelerators are usually orders of magnitudes faster than the GPPs for neural network processing especially large convolution layers, we can focus on the last few small layers to ensure negligible performance loss. This constrain greatly reduces the search space of the critical layers.

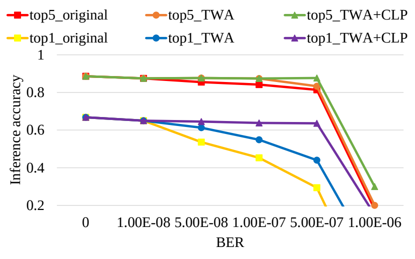
III. EXPERIMENTS

In this section, we evaluate the proposed resilient neural network training framework for accelerators with computing errors. We experimented using Caffe on a desktop computer with Intel(R) Core(TM) i7-6700 CPU @3.40GHz and 32GB memory. The computing errors can be caused by various relaxed design constraints and we used random computing errors in the experiments for general analysis. We injected random bit errors to input/intermediate/output features and weights as well as hidden layer status of neural networks. 8bit fixed point representation was used through the experiments. The error injection is measured with bit error rate (BER) according to [9]. In addition, we also had random errors injected to the internal computing results. To evaluate the training, we take AlexNet, VGG-16 and VGG-19 as the benchmark. The analysis can be applied to more neural networks.

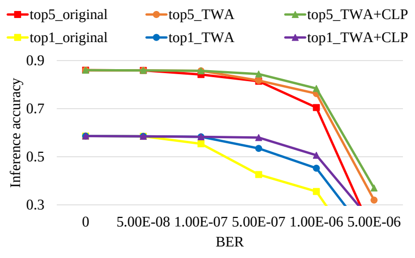
To explore the resilience of the proposed neural network training, we compare the prediction accuracy of neural networks in three scenarios. 1) We have the offline trained neural network models deployed on CNN accelerators with computing errors directly. This case is denoted as 'original'. 2) We have the neural network models retrained on the accelerator with computing errors. It is represented as training with accelerator (TWA). 3) We have the critical layers protected by offloading them to reliable GPPs. Then we perform the retraining. It is denoted as critical layer protected(TWA+CLP).



(a) AlexNet



(b) VGG-16



(c) VGG-19

Fig. 3. The precision accuracy of the benchmark neural network models on accelerators with different computing errors. The neural network model is meaningful when the prediction accuracy is still acceptable. For the three models, we are only interested in situations when the BER is 1E-7, 5E-7 and 5E-7 respectively.

The comparison of the three cases is presented in Figure 3. When the BER goes up, the prediction accuracy of the original neural network drops considerably despite the resilience of the neural networks. With the proposed training i.e. TWA+CLP, the top1 and top5 precision accuracy of the retrained models improves by 20.7% and 5.9% on average respectively compared to the offline trained model at the extreme yet acceptable error injection rate. The great prediction accuracy improvement indicates that the resilience of the retrained neural network models is improved targeting at the specific

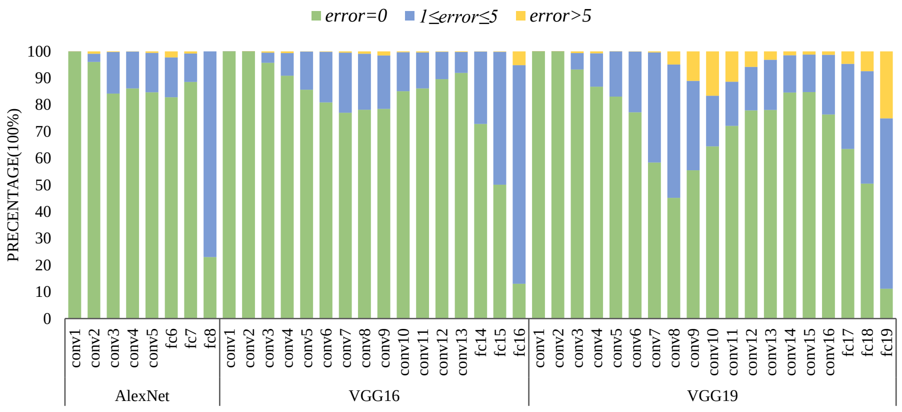


Fig. 4. Error distribution across the neural network layers when highest BER is used in AlexNet, VGG16 and VGG19.

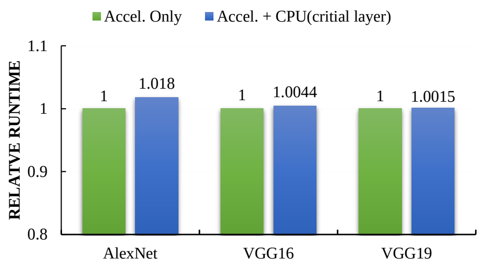


Fig. 5. Relative runtime of neural networks when the critical layer is scheduled to CPU.

computing error pattern. Therefore, more aggressive design trade-offs between prediction accuracy and performance or energy efficiency can be performed.

We decided the critical layers using the error distribution as shown in Figure 4. We set the error threshold to be 5 and the experiment reveals that the last FC layer has the largest portion of computing errors that are more than 5. Thus, it is considered as the most critical layer. The critical layer takes only a small portion of the overall neural network computing, so the performance penalty is small even when it is scheduled to CPU. The last FC layer in AlexNet takes up higher portion of computing, the performance penalty is relatively higher compared to VGG16 and VGG19.

While scheduling the critical layers to GPPs may lead to additional computing overhead due to the computing gap between GPP and the accelerators, we need to evaluate the performance overhead. The relative performance of the second case and the third case is shown in Figure 5. The performance penalty is less than two percent in the three neural networks. Considering the gains of the relaxed design constraints, it is usually beneficial to schedule the small neural network computing layers to GPPs.

IV. Conclusion

In this work, we propose to replace the forward computing on GPPs with accelerator computing during training and have both the computing errors and the application data learned in the neural network models. In addition, we opt to protect critical neural layers to reduce the negative influence of computing errors. With the proposed resilient neural network training, the prediction accuracy of the retrained neural network models improves significantly when computing errors appear.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China (No. 61874124), Chinese Academy of Sciences STS (No. KFJ-STS-SCYD-226), European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement (No. 799066).

References

[1] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 110:1–110:6.

[2] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, "Caffeinated fpgas: Fpga framework for convolutional neural networks," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec 2016, pp. 265–268.

[3] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 267–278.

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2016.

[5] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb 2010.

[6] Y. Pu, X. Zhang, J. Huang, A. Muramatsu, M. Nomura, K. Hirairi, H. Takata, T. Sakurabayashi, S. Miyano, M. Takamiya, and T. Sakurai, "Misleading energy and performance claims in sub/near threshold digital systems," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 625–631.

[7] K. Shi, D. Boland, and G. A. Constantinides, "Accuracy-performance tradeoffs on an fpga through overclocking," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, April 2013, pp. 29–36.

[8] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and -1," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2014, pp. 1–6.

[9] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, June 2018, pp. 1–6.