Error Resilient Deep Neural Networks using Neuron Gradient Statistics

Chandramouli Amarnath, Mohamed Mejri, Kwondo Ma and Abhijit Chatterjee
Department of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
`chandamarnath@gatech.edu, mmejri3@gatech.edu, kma64@gatech.edu, abhijit.chatterjee@ece.gatech.edu`

Abstract

Deep neural networks (DNNs) have been widely adopted in daily life with applications ranging from face recognition to recommender systems. However, the specialized hardware used to run these systems is vulnerable to errors in computation that adversely impact accuracy. Conventional error tolerance methods cannot easily be used here due to their substantial overhead and the need to modify training algorithms to accommodate error resilience. To address this issue, this paper presents a novel approach taking advantage of the statistics of neurons' gradients with respect to their neighbors to identify and suppress erroneous neuron values. The approach is modular and is combined with an accurate, low-overhead error detection mechanism to ensure it is used only when needed, further reducing its effective cost. Deep learning models can be trained using conventional algorithms and our error correction module is fit to a trained DNN, achieving comparable or superior performance relative to baseline error correction methods. Results are presented with emphasis on scalability with regard to dataset and network size, as well as different network architectures.

1 Introduction

The increasing use of Deep Neural Networks (DNNs) in safety critical applications such as autonomous driving [1] has drawn attention to their vulnerability to soft errors [2]. To achieve high accuracy in these tasks, DNNs rely on a large number of Multiply-Accumulate (MAC) and activation operations for each input [3]. For safety critical systems such as autonomous driving, these operations are required to be performed with high accuracy in the presence of computation (soft) errors which change computation output values by inverting bits during computation [4]. These soft errors can be caused by radiation, voltage transients in device operation and alpha particle strikes. To combat them, low-overhead, low-impact online methods for error correction must be built to provide resilience against errors that may impact DNN accuracy [5].

Soft error resilience in DNN activations is addressed in [6]; erroneous activations are identified using correlations in activities of neighboring neurons and suppressed (by setting to zero), requiring modification of the DNN training process and specialized hardware for on-line error resilience. Resilience-aware computation scheduling, exploiting fault-tolerant device parts for sensitive computations has been explored in [7], allowing a flexible tradeoff between reliability and efficiency. However, this assumes a subset of processing units can be hardened against single-event upsets and may fail under high error rates. The use of median filtering (median feature selection), involving training with median filtering layers after each DNN layer computation block has been explored in [8]. This provides high resilience to soft errors and incurs low computation overhead, but requires alteration of DNN training to ensure that accuracy is unaffected by the median filter layers.

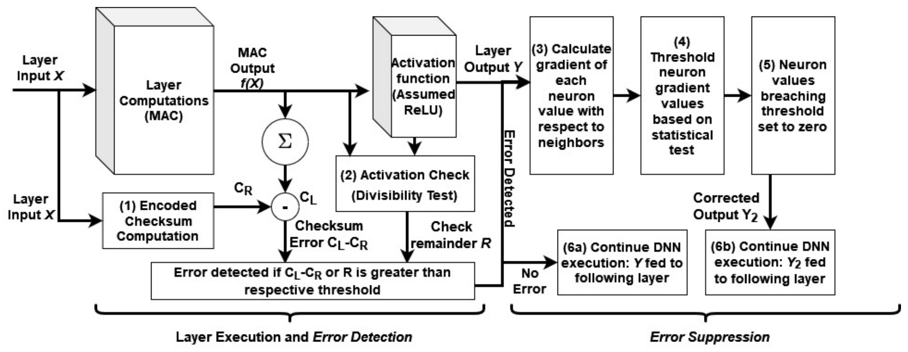

Layer Input X
Layer Computations (MAC)
MAC Output f(X)
Σ
Activation function (Assumed ReLU)
Layer Output Y
(3) Calculate gradient of each neuron value with respect to neighbors
(4) Threshold neuron gradient values based on statistical test
(5) Neuron values breaching threshold set to zero
Layer Input X
(1) Encoded Checksum Computation
C_R
C_L
-
(2) Activation Check (Divisibility Test)
Checksum Error C_L-C_R
Check remainder R
Error Detected
Corrected Output Y_2
(6a) Continue DNN execution: Y fed to following layer
(6b) Continue DNN execution: Y_2 fed to following layer
No Error
Error detected if C_L-C_R or R is greater than respective threshold
Layer Execution and Error Detection
Error Suppression

Figure 1: Proposed error resilience framework: Concurrent error detection is performed using encoded checks on DNN operation, and the presence of an error triggers the suppression system. Neurons with anomalous gradients (determined by a statistical test based on trained DNN outputs on training data) have their values set to zero before DNN computation continues.

Recent work using an opportunistic parity bit achieves high coverage detection and suppression of bit errors in DNN weights with zero storage overhead, but can be masked and does not cover activation errors [9]. Ranger [10] uses selective range restriction of DNN layer computations with restriction ranges derived from DNN performance on training data. This reduces the severity of critical faults that affect significant bits in DNN operation with low overhead. However, Ranger incurs a tradeoff between resilience and error-free accuracy that systems involving error detection followed by correction (such as our proposed approach) do not.

In contrast to prior work, extending work done in [11], this paper uses the difference of neuron outputs with respect to their neighbors (neuron outputs' gradients with respect to their neighbors) to localize and suppress errors. The proposed approach achieves *comparable or superior performance* to state of the art methods such as Ranger [10] on benchmark DNNs. Unlike prior work [6], [8], the proposed approach *does not require modification of DNN training or hardware* and can be combined with error detection mechanisms [12] to reduce the effective computational overhead. Extending [11], this paper proves the scalability of the presented method and tests it against more recent baselines such as [10]. We examine the effects of (a) scaling dataset sizes, (b) different network architectures, (c) scaling network sizes and (d) increasing the number of classes within an identically sized dataset. As with prior work, we consider error resilience in image classification DNNs.

2 Approach Overview

The presented error resilience framework (shown in Fig. 1) is composed of two steps: **(1)** *Error detection*, which is used to trigger **(2)** *Error Suppression* when an error is detected. This reduces the memory access and compute overhead compared with an "always-on" error resilience approach. This section begins with a brief overview of the error detection setup followed by an overview of the error suppression system that forms the main contribution of this research. In Fig. 1, layer operations produce an output $f(X)$ from the layer input X and the activations (assumed ReLU) produce a final output Y. These are used for concurrent error *detection* and *suppression*.

Error detection in DNN layer computations is performed in two phases, shown in Blocks 1 and 2 of Fig. 1. *First*, for the MAC computations (weight-bias multiplication or convolution) of a DNN layer, encoding methods similar to [13] (Block 1 of Fig. 1) are used for error detection. These methods produce a checksum value (C_R in Fig. 1) by encoding the DNN weight matrices or convolutional kernels. This is compared against the weighted sum of the layer outputs (C_L), which by design is equal to C_R under error-free operation. If the absolute value of $C_L - C_R$ exceeds a threshold, an error is flagged. The *second* step (Block-2) involves error detection in DNN activations (in this work assumed to be ReLU). The ReLU formulation is $y = max(0, x)$. Under ideal conditions the ReLU output y is perfectly divisible by x. If y is not perfectly divisible by the activation input x (leaves a remainder $R \neq 0$ exceeding a threshold) an error is flagged in DNN activation.

Error suppression is invoked if the error detection systems flag an error and forms the *core contribution* of this work. In Fig. 1 this is the block where $C_L - C_R$ and R are compared with their respective thresholds to check for errors. If no error is present, the DNN computations continue by passing the layer output Y to the following layer (Block 6a). If an error is present, error suppression begins by processing Y in Block-3. Here the gradient of each neuron's output value (each value in the output Y) with respect to the corresponding value of neighboring neurons is computed. *Neighboring neurons* of a given neuron are defined as adjacent neurons in a circularly ordered list of neurons in the same layer defined in the model's code. For dense layers, the gradient is the difference between neighboring neuron outputs. For convolutional layers, it is the difference between corresponding pixel values in the convolutional kernel output. This gives a set of values of the same dimension as the DNN layer output, which are sent to Block-4 for diagnosing erroneous neuron outputs. The gradient values are compared with predetermined thresholds for diagnosis (via Student's t test [14]). These thresholds are obtained from the trained DNN across the training dataset, using Welford's algorithm [15] to calculate a running mean and standard deviation for the inter-neuron gradients (differences).

Due to the 1-D convolutional kernel used to calculate the gradient (here a length-3 kernel $[-1, 1, 0]$), potentially erroneous values occur in groups for each error. The first element of this group will be the gradient of the erroneous value with respect to its clean neighbors. Anomalous gradients deemed to take place from compute errors in the corresponding neuron are corrected by setting that neuron value (or pixel value, for convolution outputs) to zero in Block-5, producing a modified layer output Y_2 that is sent on for the following layers in Block 6b. This method can be applied to all but the last DNN layer, similar to [8] and [10]. *The use of gradient values allows more flexible thresholds that accommodate a larger range of values of neuron outputs than the use of just neuron values.* This approach is validated below.

3 Experimental Results

Error injection experiments into PyTorch [16] operations in CUDA were conducted using the PyTorchFI tool [17]. PyTorch fault injection was conducted on two major image classification architectures: ResNet [18] and VGG [19]. To test the effects of scaling the number of classes, ResNet-18 and VGG-16 were both tested for error injection using the CIFAR-10 and CIFAR-100 datasets [20]. To test the effects of network size scaling, we conducted error injection simulations into ResNet-34 trained on the Tiny-ImageNet [21] dataset. The proposed framework is compared against the baseline of Ranger (output range restriction) [10] as well as the case of a network with no error correction mechanisms.

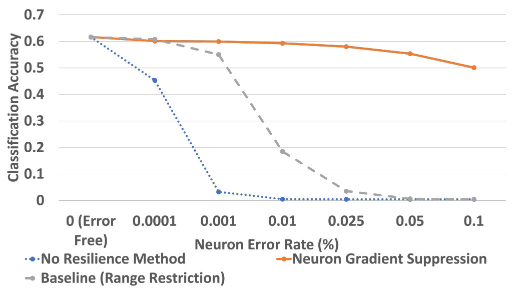
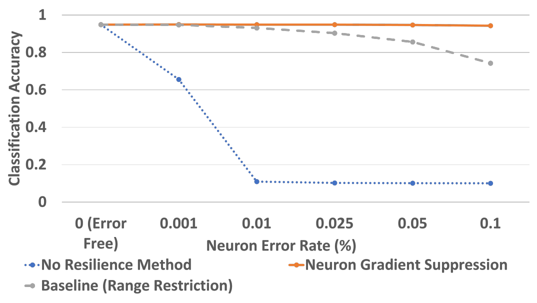


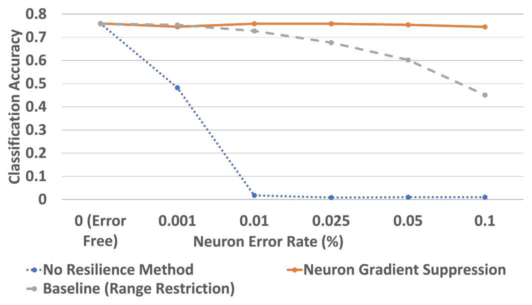
Figure 2: Results of error injection experiments on ResNet-34 trained on the Tiny-ImageNet dataset. The gradient approach achieves superior results to the other approaches, maintaining accuracy under high error rates.

Error resilience is measured using the test accuracy of the DNN under soft errors injected into the output values of neuron MAC computations. Each neuron in the DNN has a given probability of error for each inference, referred to as the *error rate*. Errors are injected using Ranger, using the proposed framework and using no error correction method. These soft errors are modeled as bitflips in 32-bit floating point DNN computations. In ResNet this was simulated by flipping a random six bits in the computation output. Soft errors are injected as *word* errors in VGG-16, where a one or more bits centered around a random position flip together. Word errors are injected such that every bit adjacent to an erroneous bit has a high probability (here 75%) to flip and be erroneous, potentially resulting in multiple erroneous bits.

Error Injection Experiments: Error injection in ResNet-18 on the CIFAR-10 and CIFAR-100 datasets is shown in Figures 3a and 3b respectively, while similar error injection experiments for VGG-16 on the CIFAR-10 and CIFAR-100 datasets are shown in Figures 4a and 4b respectively. We can see that the gradient-based error suppression approach presented here achieves slightly superior (VGG-16) and much better (ResNet-18) results on both datasets compared to the baseline (Ranger) and the network without error correction. VGG-16 shows more resilience to errors and better performance using Ranger due to the network architecture allowing a narrower range of values for each neuron. Ranger forcing values to clamp to threshold values thus does not cause misclassification.

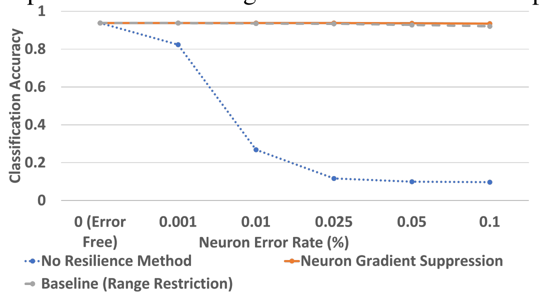


(a) Results of error injection for the CIFAR-10 dataset.

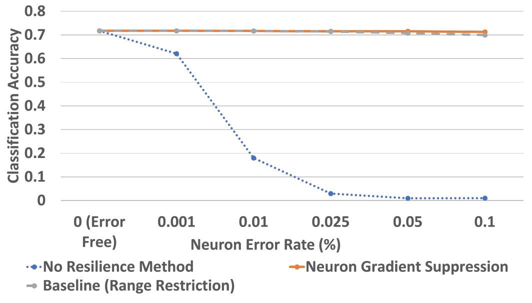


(b) Error injection experimental results for CIFAR-100.

Figure 3: Results of error injection experiments on ResNet-18. The gradient approach achieves superior results to Ranger and maintains network performance under error.



(a) Results of error injection for the CIFAR-10 dataset.



(b) Error injection experimental results for CIFAR-100.

Figure 4: Results of error injection experiments on VGG-16. The gradient approach achieves slightly superior results to Ranger and maintains network performance under error.

ResNet-18 allows a wider range of values for neuron outputs, and errors forced to clamp by range restriction can still cause misclassification, while gradient-based error suppression prevents this. It was found that gradient based error suppression was most effective at a t-test threshold large enough to approach the maximum gradient value.

Scaling Network and Dataset Size: Figure 2 shows results for similar error injection experiments on ResNet-34 trained on the Tiny ImageNet dataset, a larger dataset (200 classes, 100k training images) compared to the CIFAR10 and CIFAR100 (10 and 100 classes respectively) datasets. ResNet-34 is also a larger network in the same family as the ResNet-18 used in Figures 3a and 3b. We can see that while Ranger shows worse performance and the network without error correction mechanisms sees noticeable degradation at an order of magnitude lower error rate than the case in Figure 3b, the gradient based suppression system is able to maintain accuracy for rising error rates.

Scaling Number of Classes: Similarly, we can see from Figures 3a and 3b for the ResNet family and Figures 4a and 4b for the VGG family that raising the number of classes from 10 to 100 while keeping the number of training images fixed at 50K has little effect on the effectiveness of the proposed gradient-based zeroing approach.

4 Conclusion

This work presented an approach to DNN error resilience that enables safe operation under high error rates in hardware that outperforms the baseline while not requiring alterations to DNN training or hardware. In future we plan to implement this approach for benchmark DNNs on FPGA hardware, providing accurate measurements of hardware-level overhead, and study the effects of pruning, with its associated loss of network redundancy, on the gradient-based zeroing approach.

Acknowledgements

This research was supported by the U.S. National Science Foundation under Grant No. 2128149.

References

[1] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[2] J. Wei, Y. Ibrahim, S. Qian, *et al.*, "Analyzing the impact of soft errors in vgg networks implemented on gpus," *Microelectronics Reliability*, vol. 110, p. 113 648, 2020, ISSN: 0026-2714. DOI: `https://doi.org/10.1016/j.microrel.2020.113648`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S002627141930914X`.

[3] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017. DOI: `10.1109/JPROC.2017.2761740`.

[4] ISO, *Road vehicles – Functional safety*, Norm, 2011.

[5] B. Reagen, U. Gupta, L. Pentecost, *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6. DOI: `10.1109/DAC.2018.8465834`.

[6] E. Ozen and A. Orailoglu, "Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*, IEEE, 2020, 75:1–75:9. DOI: `10.1145/3400302.3415680`. [Online]. Available: `https://doi.org/10.1145/3400302.3415680`.

[7] C. Schorn, A. Guntoro, and G. Ascheid, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 979–984, 2018.

[8] E. Ozen and A. Orailoglu, "Boosting bit-error resilience of dnn accelerators through median feature selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 1–1, Nov. 2020. DOI: `10.1109/TCAD.2020.3012209`.

[9] S. Burel, A. Evans, and L. Anghel, "Zero-overhead protection for cnn weights," in *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2021, pp. 1–6. DOI: `10.1109/DFT52944.2021.9568363`.

[10] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 1–13. DOI: `10.1109/DSN48987.2021.00018`.

[11] C. Amarnath, M. Mejri, K. Ma, and A. Chatterjee, "Soft error resilient deep learning systems using neuron gradient statistics," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2022, pp. 1–7. DOI: `10.1109/IOLTS56730.2022.9897815`.

[12] C. Amarnath, M. I. Momtaz, and A. Chatterjee, "Addressing soft error and security threats in dnns using learning driven algorithmic checks," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2021, pp. 1–4. DOI: `10.1109/IOLTS52814.2021.9486685`.

[13] E. Ozen and A. Orailoglu, "Sanity-check: Boosting the reliability of safety-critical deep neural network applications," in *28th IEEE Asian Test Symposium, ATS 2019, Kolkata, India, December 10-13, 2019*, IEEE, 2019, pp. 7–12. DOI: `10.1109/ATS47505.2019.000-8`. [Online]. Available: `https://doi.org/10.1109/ATS47505.2019.000-8`.

[14] E. L. Lehmann and J. P. Romano, *Testing statistical hypotheses* (Springer Texts in Statistics), Third. New York: Springer, 2005, ISBN: 0-387-98864-5.

[15] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962. DOI: `10.1080/00401706.1962.10490022`. eprint: `https://www.tandfonline.com/doi/pdf/10.1080/00401706.1962.10490022`. [Online]. Available: `https://www.tandfonline.com/doi/abs/10.1080/00401706.1962.10490022`.

[16] A. Paszke, S. Gross, F. Massa, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[17] A. Mahmoud, N. Aggarwal, A. Nobbe, *et al.*, “Pytorchfi: A runtime perturbation tool for dnns,” in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 25–31.

[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[19] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *International Conference on Learning Representations*, 2015.

[20] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Tech. Rep., 2009.

[21] Y. Le and X. S. Yang, “Tiny imagenet visual recognition challenge,” 2015.