Ares: A framework for quantifying the resilience of deep neural networks

Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough
Sae Kyu Lee, Niamh Mulholland, David Brooks, Gu-Yeon Wei
Harvard University

ABSTRACT

As the use of deep neural networks continues to grow, so does the fraction of compute cycles devoted to their execution. This has led the CAD and architecture communities to devote considerable attention to building DNN hardware. Despite these efforts, the fault tolerance of DNNs has generally been overlooked. This paper is the first to conduct a large-scale, empirical study of DNN resilience. Motivated by the inherent algorithmic resilience of DNNs, we are interested in understanding the relationship between fault rate and model accuracy. To do so, we present *Ares*: a light-weight, DNN-specific fault injection framework validated within 12% of real hardware. We find that DNN fault tolerance varies by orders of magnitude with respect to model, layer type, and structure.

ACM Reference Format:
Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough and Sae Kyu Lee, Niamh Mulholland, David Brooks, Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *DAC '18: DAC '18:The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3195970.3196083

1 INTRODUCTION

Today, deep neural networks (DNNs) are being deployed at all computing scales—from energy-constrained IoT devices to TCO-optimized data centers. Given the complexity and growing scale of DNN deployment, work has mainly focused on optimizing performance and energy efficiency. This paper explores and quantifies the inherent resilience of DNNs to hardware-level faults, opening up new directions for design and optimization strategies.

Hardware is not perfect, but conventional computing systems typically require near-perfect execution to guarantee correctness. These systems are brittle and typically require fault rates on the order of 10^{-15} [1]. This level of correctness levies a high design cost at both the device and microarchitecture levels. However, relaxing these requirements can enable significant savings, and DNNs are known to be far more fault tolerant than conventional workloads (e.g., 10^{-4} [2]).

Ample opportunities to improve performance and efficiency are possible if the requirements for absolute correctness can be relaxed. These opportunities motivate us to thoroughly understand the extent of resilience in order to safely push the limits of DNN hardware from architectures to devices. At the architecture level, conventional devices can be pushed to their operational limits by increasing clock frequency to boost performance or reducing V_{dd} to maximize energy efficiency. DNNs require substantial amounts of memory to store weights and intermediate values. While modern high-density storage solutions, e.g., flash memories, require expensive redundancy and error correction hardware, such costly safeguards may be overkill for DNNs. At the device level, many emerging technologies promise high efficiency but have seen little adoption due to their instability and poor yield. At the fab level, design rules may be aggressively pushed to maximize densities at the expense of reliability.

In this paper, we present a DNN-specific fault injection framework: *Ares*. Ares studies faults at the application level and enables us to execute DNNs directly on GPUs, which is necessary for conducting large-scale fault studies. Traditional fault injection frameworks based on dynamic instrumentation can experience kernel slowdowns of up to 488× [3], making them prohibitively slow to sweep many fault patterns across a range of fault rates that vary by orders of magnitude. Ares is an accurate, first-order design space exploration framework that can guide lower-level tools on where to conduct detailed microarchitectural fault analysis. Additionally, this paper's characterization of fault tolerance at the application level can be broadly applied across a range of devices and architectures that run DNNs. This is vital because DNNs run on CPUs, GPUs, and accelerators, and the optimal microarchitecture for each DNN hardware device is still widely debated.

This paper makes the following contributions:

- We develop *Ares* to be a fast, scalable fault injection framework that enables rapid fault analysis of fully connected (FC), convolutional (CNN), and gated recurrent unit (GRU) based DNNs. Ares has been validated to be within 12% of a fabricated DNN accelerator [4].
- DNNs offer varying opportunities for optimization as model-specific, more-aggressive data types can provide 10× more resilience and fault tolerance can vary across models by several orders of magnitude.
- Abundant optimization opportunities exist when considering faults at a finer, *per-layer* granularity. Within a DNN, the per-layer fault tolerance can vary by up to 2781×. We also find that increases in fault rate translate to graceful degradation in accuracy for FC and GRU layers, and less so for CNN layers.
- We find that the different structures of DNNs (i.e., weights, activations, and hidden state) also exhibit different fault tolerance behaviors. Experiments show that the activations of a model can be up to 50× more resilient than the weights.

DAC '18, June 24–29, 2018, San Francisco, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196083

2 BACKGROUND

Deep Learning: Deep learning is a field of machine learning that leverages large neural networks to tackle challenging classification and regression tasks. DNNs have two operating modes: training and inference. Training is the process of fitting neural network parameters (weights) to labeled data. Inference is the process of using a previously trained DNN to predict labels for new input data. We can characterize the accuracy of trained models by performing inference on a particular set of previously unseen inputs. For the purposes of our study, we assume training is a large, one-time cost that has been performed in a fault-free environment. Trained models are deployed and used repeatedly, so we focus on the execution of inference.

For a more thorough discussion of deep learning, see [5].

Algorithmic Resilience of DNNs: Previous work from the machine learning community suggests DNNs can be robust to faults. It has been shown that eliminating individual nodes or parameters leads to a graceful degradation in model accuracy [6]. Furthermore, many regularization techniques can also be seen as a form of DNN robustness because correct operation requires only a fraction of the original weights. The ability to operate correctly with perturbed parameters is further supported by the large corpus of work in simplifying over-parameterized DNNs [7]. However, in these cases, individual weights or even whole neurons are removed, which is not how faults typically manifest in hardware. Instead, this paper explores bit-level fault tolerance of DNNs.

The hardware community has also shown interest in understanding DNN fault tolerance. For automotive applications, transient faults can lead to problematic image misclassifications; in order to meet ISO standards, techniques to improve reliability are required [8]. In [9], it is shown that multiple circuit faults can be circumvented with retraining in neural networks that have dozens of parameters. Finally, Minerva [2] proposes fault mitigation techniques to reduce SRAM supply voltage in order to save energy while preserving inference accuracy in fully connected DNNs.

3 THE ARES FRAMEWORK

In this section, we review fault types and sources, present Ares, and validate it against a fabricated DNN accelerator.

3.1 Faults and DNNs

Hardware Faults: Hardware designers have been forced to consider hardware reliability since the inception of the field [10]. There are many sources of faults in modern semiconductor chips and storage media. Manufacturing process variation, voltage noise, and temperature all contribute to unpredictable circuit delays that force designers to use worst-case margins. Any attempts to minimize these wasteful margins by recovering performance or power efficiency risk the occurrence of timing violation faults under certain operating conditions. SRAM circuits are also exposed to large delay variation because they use smaller devices than logic cells, and there are a vast number of cells on each chip, which increases the chance of an outlier. Similarly, heavily-scaled DRAM and flash memory cells now store so few electrons to encode each bit that they are occasionally flipped by common noise events. Cosmic particle strikes are a concern for some applications, as in datacenters, because they result in single-event

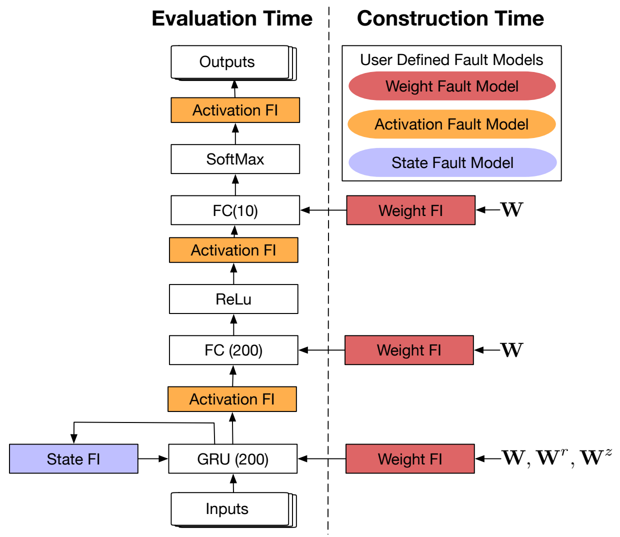

Figure 1: An illustration of the Ares fault injection (FI) framework applied to an example DNN (TiGRU).

upsets. Datapaths can also fault when operating conditions introduce excess delay.

This paper focuses on static faults in memory, which are pertinent to DNNs due to the large storage requirement for weights and intermediate states. Faults can be further classified into transient and static varieties. Transient faults come and go over time and are caused by abnormal conditions or events (e.g., resonant supply voltage noise and particle strikes). Static faults persist in the affected device and occur in cases such as “weak” SRAM bit cells due to process variation or flash life-time wear problems. Because static fault persist in time, we see them as a superset of transient faults, and studying their effects first provides a lower bound on DNN fault tolerance.

DNN Fault Points: At the algorithmic level, a DNN has four major sources of faults. Within DNN inference, there exist three sources of memory consumption where faults can occur: weights, activities, and hidden states. Faults can also occur in the datapath, which is dominated by MAC units in the case of DNNs. In this paper, we consider memory faults in all three memory fault points. Ares could be extended to study datapath faults, as all inputs and outputs to MAC units are weights, activities, and hidden states. A fault model for a MAC unit can be derived from appropriately manipulating the data before and after MAC units.

3.2 The Ares fault injection framework

Given a DNN, Ares establishes a baseline classification error by executing inference with floating point data types for all structures. Next, structures are quantized to the desired fixed point representation and the model is re-executed. For the purposes of this paper, we do not quantize models beyond the point of accuracy loss; we use the minimal fixed point type where no accuracy loss was observed.

Ares has two modes of fault injection: static and dynamic. Static faults are injected off-line, before the inference is executed. Injecting faults statically is preferred as it introduces no performance overhead. Dynamic faults are injected during the execution of a DNN inference. In Ares, the overheads of dynamic fault injection are minimized by

Table 1: *Input dimensions* vary for each dataset: *MNIST* and *CiFar-10* are 2D pixel arrays, *ImageNet* is 3D with RGB channels, and *TIDIGITS* is a time series with 2D input: the number of features per timestep by the timesteps per series. *Data type* is the minimum number of bits required without increasing error (notation is number of <integer.fraction> bits).

Model Name	LeNetFC	LeNetCNN	CF-VGG	VGG16	ResNet50	TiGRU
Dataset	MNIST	MNIST	CiFar10	ImageNet	ImageNet	TIDIGITS
Layers	3	4	12	16	54	3
Input Dimensions	28×28	28×28	32×32	224×224×3	224×224×3	39×254
Output Classes	10	10	10	1000	1000	10
Total FC Weights	270K	9.5K	270K	120M	2M	10M
Total CNN Weights	-	590K	7.6M	15M	23M	-
Total GRU Weights	-	-	-	-	-	143K
Total Activations	410	45K	250K	15M	10M	200K
Classification Error	1.68 %	0.9 %	10.17 %	35 %	31 %	3.6 %
Weights Data Type	2.6	2.8	2.10	2.10	3.13	2.12
Activation Data Type	5.3	4.5	3.12	13.4	7.6	4.4

using native tensor operations to emulate fault behavior. So long as the fault model under study can be cast as an element-wise operation (e.g., a bit flip in a matrix) or a linear transformation of the state (e.g., random or systematic noise), it can be implemented as a tensor operation and run on a GPU.

Ares performs fault injection at designated points across the weights, activations, and hidden states, depending on the DNN topology and planned experiment. Each fault injection experiment requires the bit error rate and faulty structures to be specified. Fault patterns are generated by sampling a uniformly random distributed process, which identifies which bits will fault in each structure. Ares models faults on memory structures in DNNs using bit-level fault injection to mutate values. By sweeping fault rates, the user can then analyze the fault tolerance of the DNN.

To further illustrate how Ares performs fault injection, Figure 1 lays out the network topology and fault injection points for TiGRU: a three layer DNN with one GRU and two FC layers. Ares is built on top of Keras [11], which takes high-level DNN descriptions specified in Python and executes them using either Theano [12] or TensorFlow [13] back ends. All Ares operations rely on Keras operators, which are compatible with both Theano and TensorFlow.

Fault injection is performed at two stages: construction time (static injection) and evaluation time (dynamic injection). Once the DNN is trained, the weights are known, and Ares injects weight faults at construction time by manipulating saved weights outside of Keras. In contrast, activations and hidden states are input-dependent, dynamic values. Injecting faults into these units requires changes to the Keras inference computation. Activation and state fault injection operators are implemented as GPU-compatible element-wise tensor operations. For GRU layers, we inject faults into hidden states at each time step between state updates. The network is then evaluated in the configured fault environment. We find the maximum slow down from injecting dynamic faults is less than 3.5×.

3.3 Silicon validation

To demonstrate that Ares accurately captures the bit error behavior exhibited by real hardware, we validated simulation results with measurements using a fabricated DNN accelerator capable of inducing and measuring SRAM faults [4]. The model used for the validation is the same used in the original publication: a three layer, 256 neuron-per-layer fully-connected DNN trained with *MNIST*. For silicon measurements, we sweep the scratchpad SRAM V_{dd} to control the BER for the weight storage. For each V_{dd} point, we run the entire test set to get the classification error along with the BER statistics captured by on-chip counters. Bit errors only affect the weights SRAM; the input, output, and intermediate activation data is unaffected. In Ares' simulation, the low-voltage SRAM bit errors are modeled as a uniformly distributed random process using the average BER statistics from each measured V_{dd} point, and we run the full test set to measure the equivalent simulated classification error at the same BER. Figure 2 shows the BER versus classification error curves for Ares simulation and silicon measurement.

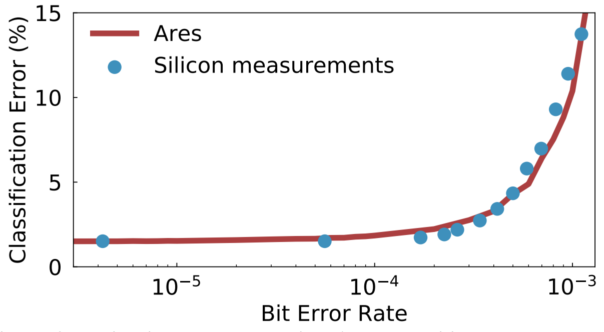

Figure 2: Validation plot comparing Ares and silicon measurements of [4] with an average error of 12%.

4 RESULTS

In this section, we use Ares to quantify the relationship between faults and accuracy in deep neural networks. We begin by presenting six DNNs that will be used in each section of the evaluation. In Section 4.2, faults are injected into the weights of each model to study how tolerance varies at the model level. Then Section 4.3 investigates injecting faults into weights one layer at a time. Finally, in Section 4.4, we compare fault tolerance across structures (i.e., weights, activities, and hidden states). All results show more than an order of magnitude variance of fault tolerance across models, between layers of models, and across structures. Hence, Ares enables significant exploration of optimizations by leveraging different aspects of resilience to faults.

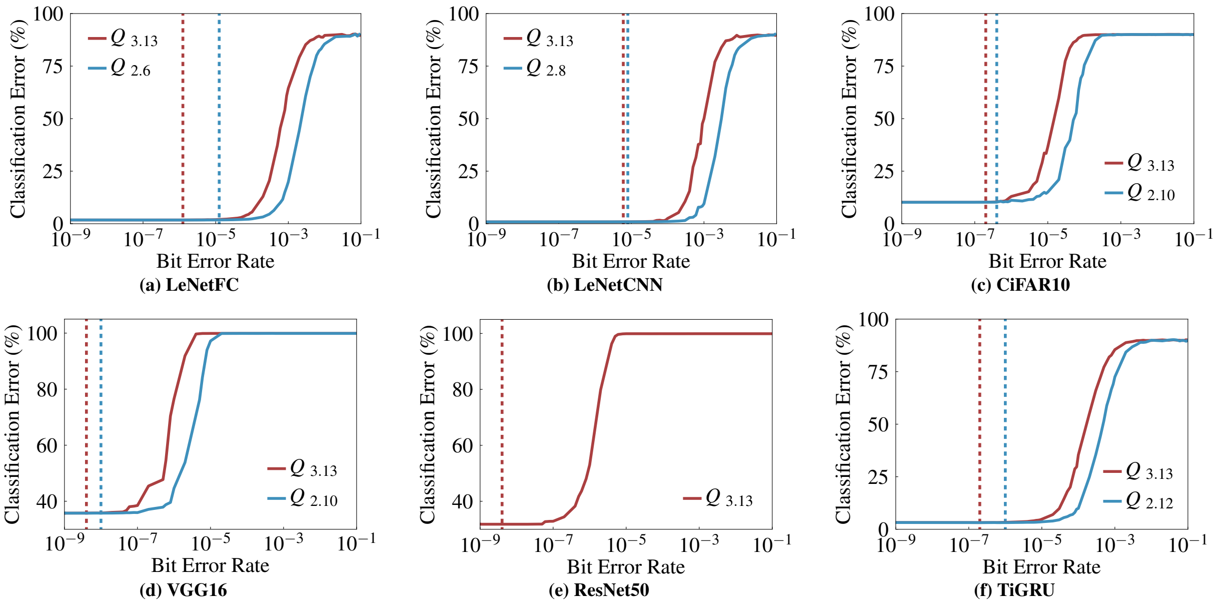

Figure 3: Bit error rate (BER) sensitivities when faults are injected in all the weights of each network. Vertical dashed lines indicate the highest BER at which there is no increase in classification error relative to the model's baseline accuracy.

4.1 Models

We have assembled a collection of six diverse DNN models to use with Ares. The datasets used to train our networks are representative of major application domains in which DNNs are commonly used: image classification (*CiFar-10*, *MNIST*, *ImageNet*) and speech classification (*TIDIGITS*). The datasets have varying sizes and the chosen models vary greatly in depth and composition. An overview of the models used is given in Table 1. For larger networks, we group clusters of similar, adjacent layers into *layer blocks* (LBs), which increases the interpretability and eases the organization when studying the deepest models.

4.2 Model-wise fault sensitivity

Starting at model granularity, we simulate the impact of bit errors by injecting faults across all the weights in each model. Figure 3 shows the resulting BER-accuracy curves, and the dashed vertical lines indicate the highest tolerable BER without loss of inference accuracy. The two vertical lines show two quantized types for each model: one using only the minimum number of bits required to run inferences without loss of accuracy (blue) and the other (red) a global type that is described below.

All models show heavily thresholded accuracy degradation: small BERs have a negligible impact on accuracy up to a certain threshold point. At BERs beyond this point, model accuracy degrades exponentially. The point of 0% accuracy loss is consistently orders of magnitude lower than the knee in the curve. Between these points, the curves typically exhibit a gradual decline in accuracy. Depending on the application, this could be a particularly interesting operating regime for devices that can tolerate some loss in accuracy in exchange for efficiency gains.

Resilience varies across DNNs. Across all models, we see a large spread in fault sensitivity; depending on the model and data type, the knee of the BER-accuracy curve can vary by multiple orders of magnitude. Considering the unified data type for all models, we see that the 0% accuracy loss can vary from 4×10^{-9} (VGG16) to 6×10^{-6} (LeNetCNN). These results suggest protection mechanisms and optimizations used (e.g., ECC level and optimal voltage reduction) should be engineered on a per-model basis.

Weight quantization impacts resilience. To improve efficiency, DNNs are quantized to minimize the total number of bits. We study weight fault tolerance with two data types for each network: a unified $Q_{3.13}$ (i.e., 3 integer and 13 fractional bits) and a per-model optimized type. $Q_{3.13}$ is used as it is the minimal type needed for no loss of accuracy across all models, with ResNet50 requiring the most bits. While a single global type would be needed to build flexible hardware that supports all models, model-specific types are the most aggressive quantizations each DNN can use without any increase to the baseline error.

Figure 3 shows that the $Q_{3.13}$ type (red) consistently results in less resilience than the model-specific types (blue). In the case of LeNetFC, the optimized $Q_{2.6}$ data type is 10× more fault tolerant. This is because the number of integer bits used (3 versus 2) determines the range of representable values, and models other than ResNet50 can be clipped to just 2 integer bits with no loss in accuracy. The unnecessarily larger range of possible values allows for faults of greater magnitude to occur, and hence increases the potential impact of a fault (e.g., flipping the MSB of a near-zero valued weight results in a greater change in value if 3 integer bits are used rather than 2). The effect of bit flips in superfluous fractional bits is not as strong. By reducing the number of integer bits from 3 to 2, model-specific quantizations consistently demonstrate increased fault tolerance.

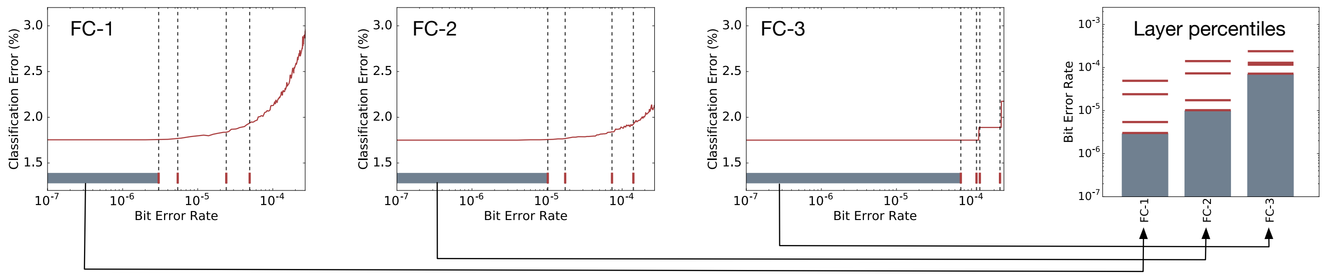

Figure 4: Per-layer LeNetFC BER vs. classification error curves (and how per-layer bar charts are derived from such curves).

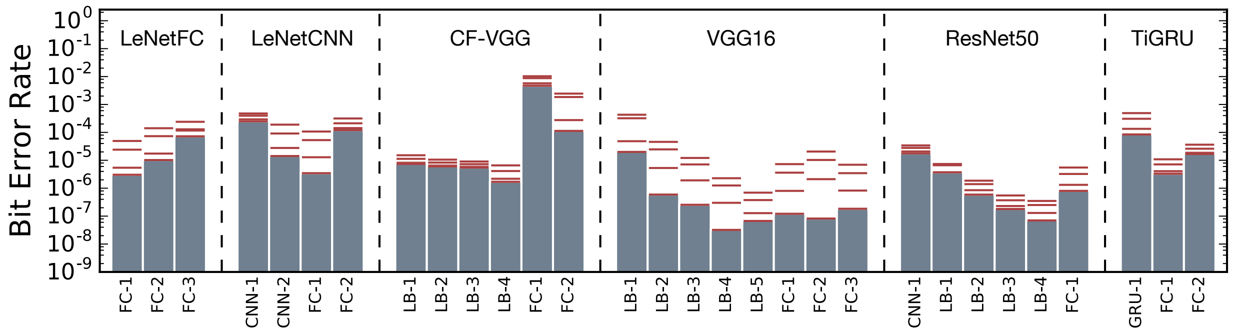

Figure 5: Per layer fault tolerance for all models. Gray bar corresponds to highest BER at which there is no measured increase in classification error. Red ticks correspond to the maximum BER for 0%, 1%, 5%, and 10% accuracy loss respectively.

Persistent class misprediction. We found the variance of the distribution of test-case mispredictions (i.e., the number of mispredictions for each class at a sampled fault pattern) is positively correlated with classification error. While fault patterns with high per-class misprediction variance are outliers, it can skew the average classification error. In LeNetCNN, 100 samples at a fault rate of 1×10^{-3} resulted in an average error of 12.2% while the median was only 6.8%. There are two ways a network can exhibit high class-misprediction variance: one class can constantly be misclassified or half the classes tend to be wrong and half correct. We found the latter case had a greater effect on model classification error.

4.3 Per-layer sensitivity

We now explore weight faults on a per-layer basis to further investigate fault sensitivities. By experimenting at per-layer granularity, we are able to understand which layers are the most resilient to faults and offer the most opportunity for hardware design optimizations.

4.3.1 LeNetFC layer fault tolerance. Figure 4 shows the per-layer BER-accuracy curves for LeNetFC. Each experiment consists of 150 different randomly sampled fault patterns at each BER. The red line indicates the mean of the sampled patterns and the blue band shows one standard deviation from the mean. At the bottom of each layer's results a horizontal gray bar highlights the maximum tolerable fault rate with zero increase in model error. The four red markers after the bar indicate the maximum fault rate with 0% (redundant with the gray bar), 1%, 5%, and 10% increase in classification error relative to the baseline error given in Table 1. The data is summarized in the bar chart to the right to clearly compare layer fault tolerances both within and across models (see Figure 5).

LeNetFC FC-3 shows distinct stepped levels. This is because FC-3 only has one thousand weights, which gives rise to discrete steps in the expected number of classification errors as the BER reaches thresholds at which one additional bit, on average, is flipped.

4.3.2 Layer sensitivities across models. Figure 5 shows the results of repeating the per-layer experiment for all models. Each bar and set of markers is constructed in the same fashion as Figure 4. The larger CNN-based models considered are very deep, e.g., ResNet50, and are grouped into layer blocks, as described in Section 4.1.

One clear takeaway from Figure 5 is that the variation in the 0% accuracy loss point between layers, both within and across models, is large. When considering a single network, the 0% point can vary between layers by as much as 2781× (LB-4 and FC-1 of CF-VGG). Across models, we find up to *five orders of magnitude difference in fault tolerance* (FC-1 of CF-VGG versus LB-4 of VGG16). Since Ares can quantify the variation of weight resilience at a fine granularity, this translates to better designs than considering faults at the model level alone.

BER-accuracy tradeoffs vary across layers. A region exists on the BER-accuracy curves between the 0% accuracy loss point and the knee where efficiency tradeoffs could be made in applications that can tolerate minor accuracy degradation. We formalize this by highlighting the 0%, 1%, 5%, and 10% points (red ticks) in Figure 5. The distance between the ticks represents the length of the tail between 0% accuracy loss and the exponential of the BER-accuracy curves. A larger distance between 0% BER and 10% relative error increase indicates a longer tail, implying a more graceful degradation in accuracy.

We find that layer blocks of VGG16 have the longest tails and ResNet50 blocks have the smallest. Tail length has implications for making BER-accuracy tradeoffs when designing hardware because it corresponds to the number of additional faults that can be tolerated without significantly affecting accuracy. A shorter tail indicates that

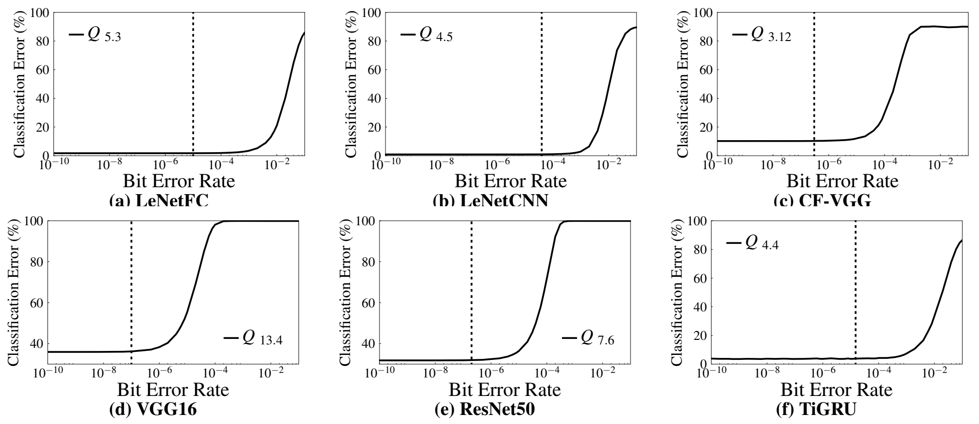

Figure 6: BER sensitivities in activation and hidden-state for GRUs for all models.

the occurrence of additional faults will push the model into the exponential region of the BER-accuracy curve, thus these layers likely require more protective margins.

4.4 Structure sensitivity

Ares can also help designers understand which structures within DNNs are the most fault tolerant. The properties of these structures have markedly different execution patterns: weights are read-only while activations and hidden state are read once then rewritten. Understanding the sensitivities of different structures to faults makes further fine-tuning and optimization possible.

Figure 6 shows BER-accuracy curves as a result of fault injection in the activation and hidden state units; weights are assumed to be fault-free to isolate the respective effects. We see that the points identified as 0% BER for activity fault injection differ by orders of magnitude in the larger models (ResNet50 and VGG16 at 2×10^{-7} and below), and smaller ones (LeNetFC, LeNetCNN, and TiGRU at 10^{-5} and above).

Activations are less sensitive than weights. For the majority of our models, we find that activations exhibit fault tolerance similar to or greater than that of the weights of the same model at an equivalent BER. Activations for ImageNet-based models (ResNet50 and VGG16) are up to 50× more tolerant to faults than their weights.

Reuse dictates sensitivity. Intuition suggests that the relatively higher resilience of activations in the largest CNN models (ResNet50 and VGG16) is due to weight reuse in CNN layers because filter kernels are convolved across the entire 2D input feature map. A fault in the weight kernel recurs and has greater impact as it is multiplied by all the activations in the associated channel, while a fault in an input activation affects a much smaller subset of values.

5 CONCLUSION

The Ares framework can rapidly and accurately quantify the fault tolerance and accuracy tradeoffs of three prominent types of DNNs: fully-connected, CNN, and GRU. Ares enables faults to be analyzed at the model, layer, and structure level, providing users with varying degrees of granularity at which to consider fault tolerance. This paper demonstrates: model-specific, more-aggressive data types can provide 10× more resilience, and fault tolerance can vary across models by several orders of magnitude; per-layer fault tolerance can vary by up to 2781×; activations can be up to 50× more resilient than weights. The findings of this paper suggest how future DNN hardware designs can leverage the implicit fault tolerance properties of DNNs demonstrated here to improve efficiency.

6 ACKNOWLEDGEMENTS

We thank Glenn Holloway for his help revising this work and the anonymous reviewers for their feedback. This work was supported in part by the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA. The work was also partially supported by the U.S. Government, under the DARPA CRAFT and DARPA PERFECT programs. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. Reagen was supported by a Siebel Scholarship.

REFERENCES

[1] "Solid state drive (ssd) requirements and endurance test method." https://www.jedec.org/standards-documents/focus/flash/solid-state-drives, 2017.
[2] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernandez-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," *ISCA*, 2016.
[3] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," *ISPASS*, 2017.
[4] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G. Y. Wei, "A 28nm soc with a 1.2ghz 568nj/prediction sparse deep-neural-network engine with 0.1 timing error rate tolerance for iot applications," *ISSCC*, Feb 2017.
[5] I. Goodfellow, Y. Bengio, and A. Courville in *Deep Learning*, MIT Press, 2016.
[6] P. Kerlirzin and F. Vallet, "Robustness in multilayer perceptrons," *Neural Computation*, 1993.
[7] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," *NIPS*, 1990.
[8] G. Li, S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," *SC*, 2017.
[9] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," *ISCA*, June 2012.
[10] B. Randell, P. Lee, and P. C. Treleaven, "Reliability issues in computing system design," *ACM Comput. Surv.*, June 1978.
[11] "Keras: The python deep learning library." http://keras.io, 2018.
[12] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," *SciPy*, 2010.
[13] "Tensorflow: An open-source software library for machine intelligence." https://www.tensorflow.org/, 2018.