Just Say Zero: Containing Critical Bit-Error Propagation in Deep Neural Networks With Anomalous Feature Suppression

Elbruz Ozen

University of California, San Diego
La Jolla, California, U.S.A.
elozen@eng.ucsd.edu

Alex Orailoglu

University of California, San Diego
La Jolla, California, U.S.A.
alex@cs.ucsd.edu

ABSTRACT

DNNs are abundantly employed in a variety of applications, including real-time systems with strict safety constraints. The consequences of errors prove disastrous in safety-critical systems, such as autonomous driving, healthcare, and industrial applications. DNNs are resilient to limited numerical perturbations yet fragile under large deviations in weights and activations. The traditional error tolerance measures fail to meet the tight design constraints of DNN processing systems due to extensive overheads or limited advantages in abundant error conditions. The algorithmic particularities of DNNs though create novel opportunities to deal with errors more effectively and economically. We revisit the two fundamental tasks in fault-tolerant system design, namely, error detection and correction, and demonstrate that the precise versions of these operations could be replaced by approximated counterparts in DNNs to deliver an extensive bit-error resilience even at high error rates while necessitating no information redundancy. We first maintain DNN accuracy even under extreme error rates by suppressing the numerical contributions of anomalous activations, eliminating any reliance on precise error correction. We tackle the problem of no redundancy error detection by establishing in training numerical associations among activations, and employing them for anomaly detection. Anomalous feature detection and suppression, performed efficiently at inference with minimal resources in a DNN accelerator, is shown to deliver significant resilience boosts while imposing neither information redundancy nor perceptible overheads.

CCS CONCEPTS

• **Hardware** → **Fault tolerance**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

fault tolerance, deep neural networks, robust machine learning

ACM Reference Format:

Elbruz Ozen and Alex Orailoglu. 2020. 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 '20)*, November 2–5, 2020, Virtual Event, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3400302.3415680>

This work is licensed under a Creative Commons Attribution International 4.0 License.

ICCAD '20, November 2–5, 2020, Virtual Event, USA

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-8026-3/20/11.

<https://doi.org/10.1145/3400302.3415680>

1 INTRODUCTION

The usage of deep learning techniques has proliferated in the last decade in numerous applications, with the potential of solving challenging real-life problems or composing exceedingly complex systems through efficient data use. The accuracy of deep learning techniques is exceedingly promising, yet each practical system comes with a unique set of requirements that must be precisely followed for the viability of the application. Autonomous driving, healthcare, and a myriad of industrial systems have been revolutionized by deep learning in the recent past; yet widespread acceptance hinges on an effective response to the challenges of safety particularly when hardware errors can spawn catastrophic consequences.

While DNNs are known to exhibit a noticeable immunity against imprecision and small numerical perturbations, the numerical influence of bit-errors could manifest in quite dramatic terms in digital systems due to the asymmetric weight of the bits in the number representations. The consequent degradation in the accuracy of DNN processing systems has been established in the literature on the topic [3, 4, 17, 21, 25–27, 33]; additional reliability measures are called for therefore, particularly for critical applications.

While fault tolerance has been well-studied in electronic component design, most research traditionally has focused on the rigidly precise rectification of fault effects, imposing non-negligible overheads which rapidly increase as the number of errors to be tackled grows. The imprecise nature of neural network computations lightens the restrictions on precision for error tolerance techniques, introducing the possibility of approximate fault tolerance. A technique that forgoes the precise restoration of the erroneous data but rather limits itself to the eradication of the most damaging parts of the error carries the potential of low-cost fault tolerance that nonetheless maintains neural network accuracy levels. The additional challenge of error location identification is traditionally handled through further redundancy, extracting even more overhead. Yet given the inherent redundancy and expressive flexibility of modern DNNs, it is possible to contemplate the integration of computational invariants at no additional cost at each layer for precise error checking. We demonstrate herein that an orchestrated combination of the techniques discussed constitutes a powerful yet extremely cost-effective approach to boost the bit-error resilience of DNN processing systems, thus delivering satisfaction of the strict constraints of safety-critical DNN applications while forgoing the use of standard but expensive alternatives.

The ability to handle fault tolerance at low cost and for large numbers of faults introduces the possibility of underpinning innovative DNN accelerator optimizations in performance and power, an area of intense research focus [32]. Aggressive hardware optimizations, such as voltage scaling and over-clocking, could deliver significant

boosts yet at the cost of increased fault manifestations. As long as the error tolerance techniques compensate for the concomitant large number of errors introduced by such design decisions, significant energy consumption reductions or performance boosts could be delivered. The superior resilience characteristics delivered by this work do not only constitute a cost-effective reliability solution for safety-critical designs but also contribute to the design of resource-efficient DNN applications at the edge.

2 RELATED WORK

Error detection and correction is an extensively studied problem in general-purpose computation. Parity and ECC (error correction codes) [6] are frequently utilized in safety-critical designs to protect the memory units. These codes are theoretically optimal in terms of information redundancy, yet they can not be applied to the execution path since the code invariants are not preserved through the arithmetic operations. Full redundancy, such as TMR (triple modular redundancy), is employed in automotive designs [1] to protect the execution path at the cost of a triplicated area and power consumption. Arithmetic codes [14] and ABFT [9] (Algorithm-Based Fault Tolerance) hold through arithmetic operations, and offer an alternative cheaper than full redundancy for arithmetic units. Nevertheless, they still impose a noticeable redundancy with concomitant extensions in the execution hardware. The Razor flip-flop [5] is an efficient technique for timing error detection, but with limited applicability to other error scenarios.

The bit-error resilience of DNNs is examined in the prior literature through error injection experiments on the DNN graph [17, 21, 25] and physical neutron beam experiments on GPUs [4]. Various error detection techniques are proposed for neural network computations, which operate through the identification of error symptoms [17, 28], or through the establishment and subsequent check of the computational invariants in the DNN operations [22, 23]. Previous work [26, 33] incorporates circuit-level error detection techniques with architectural error bypassing/masking to mitigate the effect of timing and SRAM errors in voltage underscaled DNN accelerators. The irregular resilience behavior of DNN weights and activations is noted in [3, 27], and the described characteristic is utilized to improve the resilience of neural network processing systems through reliability-aware resource allocation techniques. An error-correcting output coding scheme is utilized in [19] to enhance the self-correcting capability of neural networks. Noise training is demonstrated to be extremely helpful in boosting the resilience of neural networks against analog domain perturbations [8, 20]. Previous studies [2, 18, 34] have demonstrated that DNNs could even operate on defective hardware if the faulty components are properly contained, or the model is allowed to adapt to device particularities through additional fine-tuning with the extracted defect models. While advancing the field on various fronts, the outlined research works in this section fall short of being able to offer a versatile solution for boosting the comprehensive error resilience of DNNs against a diverse set of hardware errors.

3 METHOD DESCRIPTION

3.1 Overview

In this section, we reconsider the role of error detection and correction tasks in designing error-resilient DNN processing systems.

We first study the problem of error correction, whose implementation through traditional techniques, such as TMR [1], ECC [6], arithmetic codes [14], or ABFT [9] imposes additional overheads in terms of information redundancy and computational overheads. We will demonstrate in Section 3.2 how clamping the contribution of erroneous variables on the DNN outputs suffices in delivering superb DNN accuracy retention. Consequently, error effects can be effectively tackled in a cost-effective manner without attempting to recover the original version of the affected variables. Although a variety of techniques of differing costs could be considered to attenuate error propagation to DNN outputs, the desired outcome could be attained by merely dropping the erroneous variables, or alternatively clipping their magnitude, either of which being able to be implemented in hardware with trifling resources.

The error correction task can be dealt with in a cost-effective manner through error suppression; the issue of error detection to pinpoint erroneous weight and activation variables invites further scrutiny though. A variety of error detection techniques proposed in the prior literature could be utilized for this purpose, such as Razor flip-flops [5], parity bits, or arithmetic codes [14], if the associated information redundancy, area, power or performance overheads, remain within budget. The considerable cost of these techniques could be alleviated by noting that neither is precise error detection likely to be necessitated in DNN computations; instead, we explore the possibility of locating critical bit-errors with high-precision through numerical associations among DNN variables in Section 3.3. We first integrate anomaly detection and suppression layers into DNNs to locate anomalous variables through numerical order relationships among neighboring activations and prevent error propagation by squashing the numerical contribution of impacted variables. This technique is demonstrated to be extremely effective in detecting and preventing the propagation of both weight and activation errors if the rules are applied in a channel-wise manner in the convolutional layers. DNNs could adapt to anomaly detection and suppression layers merely by integrating such rules in training. Training DNN models with the integrated rules is more challenging due to the discontinuous nature of these layers and the consequent adverse impact on gradient propagation. We further improve the gradient flow by replacing discrete suppression rules with smoother counterparts in the back-propagation, control unbounded activations with activation regularization, and train the models in a guided two-stage process to assure a faster convergence to the compatible solution with competitive accuracy.

In Section 3.4, we explain how an efficient hardware module can perform the proposed operations in a DNN accelerator at inference time. As a result, extensive bit-error resilience could be attained in modern DNN processing systems at negligible area, power, and performance overheads.

3.2 Containing the Impact of Bit-Errors in Deep Neural Networks

Neural networks generally exhibit a tolerant response to small numerical variations (i.e., analog domain noise) in weights and activations [13, 20]. Yet, the impact of errors is shown to be asymmetric, with errors being mostly benign unless they lead to a significant numerical increase in variable magnitudes [17, 25]. We investigate this phenomenon in Figure 1, where we compare the accuracy of

Figure 1: Bit-errors vs. drop errors on ResNet-18 [7] (with CIFAR-10 [15]) (a) activation errors (b) weight errors

the identical DNN model by injecting bit-errors and drop errors into DNN variables. While bit-errors could have a variable magnitude effect, drop errors simply set the value of the impacted variable to zero. The reader will note that the DNN model exhibits a much more graceful response when the erroneous variables are dropped rather than when they suffer the effects of single-bit errors, with consequent resilience improvements of up to $1000 - 10000\times$. Previous studies confirm the resilience boosts under the drop errors [26, 33], while the relative sensitivity of DNNs to weight vs. activation errors shows variation [25, 26, 33].

The phenomenon in Figure 1 could perhaps be better explained if the algorithmic properties of DNNs are considered. First, previous studies [32] have established that the majority of DNN variables tend to cluster around zero. Such a characteristic is often boosted by certain design choices, including the common usage of Dropout [30], ReLU (rectified linear unit) activation function, and the regularization techniques. As a result, correcting a value by setting it to zero often introduces nothing but an inconsequential numerical discrepancy. Second, computation units (i.e., neurons or filters) are rife with redundancy in DNN layers [23], where the correlated units extract similar features. Thus, the drop of a particular variable rarely leads to a complete information loss of an extracted feature. Setting a variable imputed to be faulty to zero is consequently a safer option on average than allowing the bit-error to fester, although the numerical perturbation of the bit-error might be occasionally smaller than the perturbation introduced by the value drop.

The investigated phenomenon is extremely useful in practice. The erroneous values could be set to zero, or their magnitude could be restrained (i.e., clipped) to maintain accuracy even at high error rates without precise error correction, thus completely sidelining the use of information redundancy and expensive operations.

3.3 Complementing Variable Suppression With Low-Cost Error Detection

3.3.1 Locating Anomalous Activations Through Numerical Associations. Critical bit-errors are frequently associated with anomalously large numerical deviations in DNN variables. As a result, variable magnitude could be utilized as an indicator of critical errors. For instance, the expected numerical range of activations could be measured to determine numerical thresholds at each layer, and these thresholds can be utilized to locate erroneous variables as in [17].

Although the suggested technique requires minimal information redundancy, its performance strictly depends on how the numerical range of the data types is utilized at each layer. A single threshold per layer may not deliver sufficient resolution to locate all critical errors if the range of activations varies significantly within the layer. An alternative approach could be envisioned consisting of the utilization of unique threshold values for each activation, yet its significant cost in the number of additional parameters in the network dooms its practicality.

Our novel approach involves instead the utilization of numerical relationships among DNN activations to locate anomalous deviations. Relationships such as equivalence or strong correlation could be contemplated, yet such relationships are hard to satisfy and further drastically reduce the overall information content of the involved variables. On the other hand, numerical order relationships among DNN activations could deliver a sound numerical bound while still retaining flexibility to represent diverse information content. Such an approach resolves the problems mentioned earlier. First, each DNN activation is constrained with a unique bound, which leaves a smaller headroom for errors to escape detection compared to utilizing a global range for the entire layer. Second, no additional parameters are required as each DNN activation value is compared with another activation variable within the same layer.

We demonstrate this approach by combining it with activation dropping in a simple suppression rule provided in Equation 1. We consider a layer l with n_l computation units (i.e., neuron or convolution filter). In Equation 1, $A_{(c,x,y)}$ and $\hat{A}_{(c,x,y)}$ signify the c 'th unit output after the activation function and its propagated form through the suppression rule, respectively.¹ $U(i)$ is the step function where $U(i) = 1$ if $i > 0$; otherwise $U(i) = 0$. γ denotes a threshold coefficient that can be shared per layer or globally, with global sharing reducing the storage requirements commensurately. In simpler terms, the proposed rule outlined in Equation 1 propagates an activation value if its magnitude is smaller than the scaled version of its neighboring activation's magnitude; otherwise, it is dropped.² As the individual thresholds for each activation are determined through the magnitude of the neighboring activation, no additional value storage is necessitated.

¹ x and y are utilized to indicate the spatial dimensions of the output feature maps in the convolution layers and not utilized in the fully connected layers.

²Indices out of bound are rolled over to the other end of the array in Equation 1, i.e., the first unit ($c = 0$) checks the last unit's ($c = n_l - 1$) output.

$$\hat{A}_{(c,x,y)} = A_{(c,x,y)} \times U(\gamma|A_{(c+1,x,y)}| - |A_{(c,x,y)}|) \\ 0 \leq c < n_l, \quad \forall \{x, y\} \quad (1)$$

The presented suppression rule in Equation 1 is particularly strict since activation is immediately dropped after exceeding a single threshold, which is determined by the neighboring activation's value. This constraint could be relaxed further by introducing multiple thresholds derived from multiple neighboring activations' values, and carrying out the dropping decision only when the majority (or all) of the conditions are violated. For instance, the suppression rule in Equation 2 is controlled by two gate conditions, and this rule allows a significant numerical variation among two neighboring activations (e.g., an edge type variation), yet precludes the propagation of single activation spikes where activation is significantly larger than both of the neighboring values.

$$\hat{A}_{(c,x,y)} = A_{(c,x,y)} \times \\ [1 - U(|A_{(c,x,y)}| - \gamma|A_{(c+1,x,y)}|) \times U(|A_{(c,x,y)}| - \gamma|A_{(c-1,x,y)}|)] \\ 0 \leq c < n_l, \quad \forall \{x, y\} \quad (2)$$

It could be seen that the suppression rules are applied in a channel-wise manner, as indicated by the iteration variable c in Equation 1 and Equation 2. Such an application order delivers a particular advantage against weight errors as well. An impacted weight value in a neuron or convolutional filter propagates into the entire channel of activations.³ If the threshold activations are chosen from the adjacent channels, even the entire channel of anomalous activations in the output feature map could be easily filtered; thus, the weight error resilience of the network could be boosted in both fully connected and convolutional layers.

The proposed rules inhibit error propagation of critical bit-errors with a large magnitude, yet it is not guaranteed that non-erroneous activations in a pre-trained DNN model will escape unscathed through Equation 1 or Equation 2. Values that stand out will be squashed independent of whether they are erroneous or not; the consequent impact on non-erroneous activations needs to be ameliorated by incorporating these rules into the DNN model in the design phase. We follow the approach of integrating the suppression rules into DNN models through training, together with the ability to accomplish the desired classification task accurately.

3.3.2 Training DNNs Efficiently With the Integrated Suppression Rules. We perform DNN training with the suppression rules integrated after each convolutional and fully-connected layer to ensure a seamless adaptation to these operations. However, training DNN models with these rules involves particular challenges. We will describe some of these challenges and present various techniques to deal with the difficulty of training in this section. Before moving into the details of these challenges, it should be noted that the proposed rules alter the gradient flow in the backward pass. For instance, a gradient on the output activation variable $\hat{A}_{(c,x,y)}$ needs to be back-propagated to both $A_{(c,x,y)}$ and $A_{(c+1,x,y)}$ in Equation 1 since both variables take a role in the generation of $\hat{A}_{(c,x,y)}$.

³A channel consists of a single neuron output in the fully connected layers.

The main difficulties in training stem from the flat regions and the discontinuities of the step function. The issue can be clearly demonstrated through the derivation of the back-propagation rules for Equation 1 (from $\hat{A}_{(c,x,y)}$ to $A_{(c,x,y)}$, and from $\hat{A}_{(c,x,y)}$ to $A_{(c+1,x,y)}$). Let us assume a single DNN output O , then utilize the chain rule to derive the corresponding gradients. We make simplifications based on the fact that the derivative of the step function is zero everywhere except being undefined when the input is zero:

$$\frac{\partial O}{\partial A_c} = \frac{\partial O}{\partial \hat{A}_c} \times \frac{\partial \hat{A}_c}{\partial A_c} \\ = \frac{\partial O}{\partial \hat{A}_c} \times [U(\gamma|A_{c+1}| - |A_c|) + A_c U'(\gamma|A_{c+1}| - |A_c|)] \quad (3) \\ = \frac{\partial O}{\partial \hat{A}_c} \times U(\gamma|A_{c+1}| - |A_c|)$$

The back-propagation rule in Equation 3 is intuitive, as the gradient in $\hat{A}_{(c,x,y)}$ is back-propagated to $A_{(c,x,y)}$ as long as $A_{(c,x,y)}$ is not dropped; otherwise, the received gradient becomes zero. The gradient can be derived for the gate activation variable similarly as in Equation 4:

$$\frac{\partial O}{\partial A_{c+1}} = \frac{\partial O}{\partial \hat{A}_c} \times \frac{\partial \hat{A}_c}{\partial A_{c+1}} \\ = \frac{\partial O}{\partial \hat{A}_c} \times [A_c \times U'(\gamma \times |A_{c+1}| - |A_c|)] = 0 \quad (4)$$

Equation 3 and Equation 4 demonstrate that the gradient back-propagation imposes certain challenges on both input ($A_{(c,x,y)}$) and gate ($A_{(c+1,x,y)}$) activation variables. First, $A_{(c,x,y)}$ never gets updated if it always ends up being dropped. Second, $A_{(c+1,x,y)}$ receives no gradient at all through the value it has controlled; therefore, it is independently optimized despite its significant role in the generation of $\hat{A}_{(c,x,y)}$. The step function is desired in the forward pass because it effectively eliminates the propagation of the anomalous activations, yet hinders the gradient flow in the backward pass in its current form and precludes efficient training.

A similar problem has been previously tackled in previous work in BNNs (binary neural networks) to deal with the gradient back-propagation problem of the *sign* function. The sign function has zero gradients almost everywhere, except for the undefined gradient when the input is zero, as does the step function just being discussed. Hubara et al. [10] utilize the sign function in the forward pass, but they back-propagate the gradients from the sign function output to sign function input by approximating it as an *identity* function in the backward pass (straight-through estimator). In an analogous manner, we propose using a function that is structurally similar to the step function, yet exhibits better differentiability characteristics in the backward pass. The well-known *sigmoid* activation function fulfills the outlined criteria. While being structurally similar to the step function, the sigmoid gradients diminish more gradually as the magnitude of the input increases. As a result, gradients could back-propagate through the sigmoid if its input is close to zero. In sum, we utilize the step function in the forward pass, yet derive a back-propagation rule by replacing the step function with the sigmoid and enable a more efficient gradient flow.

We further observe that activation regularization usually has a positive effect on the training. An activation variable could be stuck at significantly large values and may fail to be updated through gradients if it saturates the sigmoid inputs. Adding its magnitude as a penalty term in the cost function, e.g., L1 or L2 norm of the activations (before suppression), reduces the large activation magnitudes back to the appropriate range, thus enabling their continuous updating through output gradients.

The proposed techniques improve the gradient flow and make it possible to train DNN models despite the presence of wide-spread discontinuities and plateaus that hinder the effective flow of the gradients. However, it might require more iterations to train deeper DNNs in more challenging datasets when these rules are integrated. We further significantly speed up the process by training more complex models in a two-stage process. In the first stage, we enforce certain restrictions on DNN weights to allow seamless integration with the described suppression rules, although the first stage of training is carried without explicitly embedding these rules. The restricted model is faster to train, yet the imposed restrictions also limit its degrees of freedom. As a result, the baseline accuracy of the constrained model suffers in comparison to the one that can be attained through the unconstrained model. In the second stage, we first integrate the feature suppression rules into the model, relax the enforced weight constraints, and fine-tune the network until it fully recovers the baseline model accuracy.

Let us consider the two-gate rule provided in Equation 2. Each activation needs to satisfy at least one of the criteria imposed by its neighbor activations to propagate its value into the next layer. We allocate the neighboring neurons/filters into pairs at each layer and tie the weight connections of each pair so that both neurons/filters become equivalent. As a result, each neuron/filter pair produces the same output, and they further receive matching gradients in the backward pass. Although the imposed constraint effectively reduces the information content at each layer by half, it also ensures that the trained model will work seamlessly with the suppression rule in Equation 2 at the end of the first stage. As each neuron/filter output will be repeated by at least one of its neighbors, it is guaranteed that no information content will be suppressed in Equation 2. In the second stage, we first integrate the feature suppression rules after each layer, lift the equivalence restriction on the weights, and perform additional fine-tuning to augment the information content and recover the remaining accuracy while the initially equivalent neurons/filters are being differentiated. We observe that most of the accuracy content of the model could be attained in the first stage, and the second stage could quickly bridge the accuracy gap with a small number of fine-tuning iterations while assuring seamless compatibility with the feature suppression rules.

The memory and computational requirements of the first stage could be further reduced by 50% by merely training a model with half of the size at each layer, then duplicating each neuron/filter and scaling the outgoing connections by half. This approach guarantees that the same outcome is attained at a smaller computational cost at the end of the first training stage. The overall design flow is visually demonstrated in Figure 2.

Finally, the output layer needs to be tackled differently than the other layers since the output layer is trained to produce results

Figure 2: Training DNNs in a two-stage process

resembling a one-hot encoding format. We, therefore, do not integrate these rules into the last layer in training. Instead, output units are duplicated after training, and the suppression rules are applied at inference, which guarantees that certain outputs will pass through these rules unaffected, and could be used to recover the classification result. This modification guarantees that the output layer will be protected against weight and activation errors while requiring a minimal increase in the parameters and operations.

3.4 Anomaly Detection and Suppression at Inference Time

We carry out anomaly detection and suppression operations entirely on hardware at inference time, which allows us to implement these operations with minimal hardware resources and with no consequent performance issues. First, we will look into the design of a hardware unit that can efficiently carry out the anomaly detection and feature suppression operations with minimal resources. We will then describe how these operation stages are positioned within the data flow of a typical DNN accelerator to deliver extensive coverage against a variety of bit-error cases that occur in both the accelerator buffers (e.g., weight, activation) and the computational fabric (e.g., systolic array).

The required hardware operations involve magnitude calculation, multiplication with a scaling factor (γ), comparison, and a selection circuitry (multiplexers and AND gates) for dropping or clipping the desired features. Within the basic components in the list, multiplication operations particularly stand out in terms of expense; therefore, we reduce the need for explicit multiplication operations by limiting the scaling factor (γ) into powers of two, obtainable merely through arithmetic shifting. The vectorized hardware implementation is demonstrated in Figure 3. The hardware unit first calculates the enforced thresholds on the fly by finding the magnitude of each element, then scaling with γ through arithmetic shifting where the shift amount is controlled by an external signal. The simplicity of the required operations makes the threshold calculations extremely efficient. The calculated threshold vectors are aligned with the activation magnitude vectors through element-wise shifting and numerically compared to determine if the threshold is violated for each activation value. An external signal (*Gate Count*) determines whether the suppress signal for each activation is generated by considering the violation signals for either one

Figure 3: Vectorized implementation of anomaly detection and feature suppression in hardware

or both of the thresholds calculated through the neighboring activation values. If the activation values need to be dropped, they can be replaced directly with zero. Clipping necessitates a slightly more complex maneuver in that it relies on the identification of the largest threshold value violated while preserving the sign of input activations in the suppressed values. Finally, the generated suppress signals determine if the original or suppressed values are reflected to the module output.

The proposed hardware module could be integrated into the outputs of the activation buffer, as in Figure 4. The demonstrated architecture loads weights into the computational fabric (i.e., systolic array) through its weight buffer. The processing of each layer is performed by fetching the layer inputs from the activation buffer, performing multiply-accumulate operations, accumulating the results if necessary, and finally generating the layer output by passing the results through the activation pipeline. We observe that positioning the proposed stage just before processing each layer delivers advantages against errors occurring in various locations of the accelerator. First, any bit-error in the activation buffer will be checked and suppressed before getting involved in further computations. Second, any bit-error in the computational fabric or the activation pipeline will be stored in the activation buffer and then checked prior to processing the next layer. The output layer will be protected through the introduced small redundancy when the outputs are checked by the proposed hardware unit before transferring to the host device memory. Third, the proposed channel-wise checking scheme is useful, as an error in the weight buffer will impact a single neuron output or feature map channel, and it can be effectively checked and suppressed by the additional hardware before processing each layer.

4 EXPERIMENTAL METHOD

We demonstrate the reliability improvements on three DNN models that are trained on three different datasets: LeNet-5 on the MNIST dataset [16], SqueezeNet [11] on the GTSRB dataset [31], and finally ResNet-18 [7] on the CIFAR-10 [15] dataset. The accuracies of the baseline models are 99.31%, 94.57%, and 88.63%, respectively. All models and experiments are implemented in PyTorch [24].

We design the proposed feature suppression rules as a custom DNN layer. The layer supports the definition of the number of gate activation variables. Namely, we utilize a single gate for each

Figure 4: Hardware integration into a DNN accelerator (Accelerator diagram is inspired by [12])

activation (Equation 1) in LeNet-5 and two gates (Equation 2) in SqueezeNet and ResNet-18 models. We explore the options of both feature dropping and feature clipping to suppress anomalous activations with various threshold coefficients (γ). An activation is immediately dropped upon a violation in the first option. The latter option allows the magnitude of the activation to be clipped to the violated threshold. When two gate variables are utilized, the clipped magnitude is adjusted to the largest threshold violated. The models trained with the suppression rules exhibit comparable accuracy values compared to baseline models, where the test accuracy values are within 0.2 – 0.5%, 0.3 – 0.4%, and 1.3 – 2.4% for LeNet-5, ResNet-18, and the highly irredundant SqueezeNet model, respectively. The LeNet-5 model can be trained directly with the integrated suppression rules in 100 training epochs (identical to the baseline model). The SqueezeNet and ResNet-18 models are trained in a two-stage process. We first train half-sized models with the same number of iterations required for the baseline models (100 epochs for SqueezeNet, 200 epochs for ResNet-18). We perform layer modifications, integrate the suppression rules, and perform further fine-tuning (100 epochs for SqueezeNet, 50 epochs for ResNet-18) in the second stage. The highest reported accuracy values are attained by utilizing the step function in the forward pass of the suppression layer and performing the backward pass with the sigmoid approximation. Utilizing the step function or the straight-through estimator in the backward pass often leads to instability in the training process. The straight-through estimator is oblivious to the forward pass behavior of the suppression layer, while the step function does not provide an efficient gradient flow.

We design the hardware modules in Verilog HDL, integrate them into the DNNWeaver v2.0 [29] open-source DNN accelerator design, and perform hardware synthesis with the Synopsys Design Compiler to measure the area, power, and timing overheads. On-chip buffers (i.e., input, weight, bias) are not included in the synthesized design. We utilize the Silvaco Open-Cell (15nm) and the Synopsys DesignWare libraries in the synthesis process. The target frequency is chosen as 1 GHz for both 8-bit and 16-bit designs.

Error injection is commonly utilized in both academia and industry to verify the safety claims of safety-critical designs. Error injection experiments need to be repeated exhaustively to reach

Figure 5: A comparison of bit-error resilience improvements in various DNN models

statistically significant conclusions. However, full hardware simulation of a system is computationally expensive, and runtime constraints often limit the applicability of exhaustive error injection experiments when practical designs are considered. We tackle this problem by first generating hardware-specific bit-error models, expressing them on the abstraction of the DNN graph, and performing exhaustive simulations through our error injection framework in PyTorch, at the speed of DNN inference. The pure dataflow nature of DNNs reduces the potential mismatch between hardware errors and our bit-error models, thus boosting confidence in the analysis.

Our simulation environment can inject errors into DNN weights and activations. The data types are assumed to be quantized to the 2's complement fixed-point format. We consider both 8-bit (LeNet-5) and 16-bit (SqueezeNet, ResNet-18) data types for various DNN models. The data range for each network is profiled to determine the required integer and fraction bits. After the desired error rate is provided, the framework randomly determines the erroneous variables, impacted bit positions, and performs inference to measure the accuracy with the applied perturbations. The injection is performed

before the prediction for the weight errors, and during the prediction for the activation errors. The experiments are repeated at the desired error rate until we obtain statistically consistent results.

5 EXPERIMENTAL RESULTS

In this section, we will first demonstrate the reliability improvements attained through anomalous feature detection combined with feature suppression techniques (i.e., dropping and clipping) and compare the results to other error tolerance methods. Second, we will characterize the area and power footprint of the hardware module that carries out anomalous feature detection and suppression operations when integrated into a DNN accelerator.

Figure 5 demonstrates the accuracy of three distinct DNN models separately under the weight and activation errors. We consider various versions of each model that are equipped with different error tolerance techniques. *Baseline* represents the bit-error resilience of a model with no additional safety features. *TMR* is the experimentally measured bit-error resilience of a coarse-grained triple modular redundancy scheme where three instances of the same network

are simultaneously executed, and the decision is made through majority voting. *Oracle ED + Drop* signifies the result for the case where the erroneous weight and activations could be precisely located through an oracle error detection technique and dropped by setting to zero. Other techniques utilize our *AFD* (anomalous feature detection) technique with the given threshold coefficient (γ) and tackle the errors by dropping or clipping to the enforced threshold. In Figure 5, the relative criticalities of the weight and activation errors vary among the models and further depend on the numerical range allocated to each type in the quantization schemes. The accuracy graphs are expected to demonstrate a larger accuracy drop in the simultaneous presence of both weight and activation errors and closely track the graph of the more critical error type.

The LeNet-5 model can tolerate up to **1000×** more errors with the same accuracy loss if all erroneous activations could be precisely located and dropped. Although precise detection of errors requires additional information redundancy, the *AFD* technique provides error detection for free, yet the comparable precision of our technique enables the model to tolerate up to **300×** more errors when it is coupled with suppression through feature dropping. The precision of *AFD* is similar to oracle error detection in particular networks, especially at lower error rates. *AFD* delivers resilience at up to **500×** higher activation error rates with the same accuracy loss compared to the baseline SqueezeNet model when combined with feature dropping. The error rate where accuracy loss commences (e.g., $< 1\%$) could be delayed to almost **10000×** larger error rates in ResNet-18 if the erroneous activations are precisely located and dropped. Similarly, the majority of errors could still be detected by *AFD* at no additional cost and dropped; thus, the same accuracy drop point could be delayed to $\sim$ **2000×** higher error rates compared to the base model. The superiority of feature dropping over feature clipping in terms of fault resilience for the same threshold coefficient, γ , values is countervailed when feature clipping utilizes a lower threshold coefficient; feature clipping with ($\gamma = 1$) delivers comparable results to feature dropping (with $\gamma = 2$), while retaining comparable baseline accuracy.

In addition, the impact of weight errors could be significantly inhibited without requiring precise error correction if the erroneous weights could be located with high accuracy, and set to zero before the execution. This approach is observed to provide resilience at up to $\sim$ **1000 – 3000×** more errors with the same accuracy loss when compared to baseline in Figure 5. Our technique does not directly operate on the weights but instead monitors and suppresses the activations, yet delivers particular benefits against weight errors, mainly when applied in a channel-wise manner as described in the previous sections. The anomalous feature detection combined with suppression allows the network to operate with the same accuracy loss at **20×** larger error rates in the weights than the baseline model. The benefits could be improved further by operating directly on the weights if the anomalous weights are monitored and suppressed in the buffers or before loading into the computational fabric. The investigation of this technique will be the focus of our future work.

Table 1 demonstrates the area and power footprints of the entire accelerator together with the share of the proposed hardware for 8-bit and 16-bit designs. The area footprint of the proposed hardware module is around **0.2 – 0.3%** of the entire design, a minuscule fraction of the chip area budget. Likewise, the power overheads are

Table 1: Area and power footprint of the hardware module

8-bit Design			
	Accelerator	AFD + Suppression.	%
Area (mm^2)	1.744	0.004	0.23
Power (mW)	955.5	0.9	0.09
16-bit Design			
	Accelerator	AFD + Suppression	%
Area (mm^2)	2.907	0.009	0.31
Power (mW)	1160.0	1.7	0.15

bounded to within **0.1 – 0.15%** of the accelerator’s power consumption, piling in comparison to process variation driven chip power deviations. The operations could be implemented as a pipeline stage, which seamlessly fits into the design without incurring any throughput loss, but only an extra cycle in latency. The synthesis experiments validate that the combinational delay of the proposed module fits into a single clock cycle at 1 GHz frequency and does not create a timing bottleneck.

6 CONCLUSIONS

We demonstrate that the bit-error resilience of DNNs could be dramatically improved merely by containing the numerical contribution of the erroneous variables without requiring explicit error correction. This approach is further complemented with our novel error detection technique to locate anomalous activations at the cost of no information redundancy. We propose an efficient hardware architecture to perform anomaly detection and suppression operations in a DNN accelerator with negligible area and power overheads. Our experimental analysis demonstrates that the combination of these techniques enables state-of-the-art DNNs to tolerate up to **2000×** higher error rates when compared to baseline models. The provided resilience characteristics could be employed in the safety-critical designs, or paired with aggressive design optimizations to push the efficiency limits of DNN accelerators.

REFERENCES

- [1] Massimo Baleani, Alberto Ferrari, Leonardo Mangeruca, Alberto Sangiovanni-Vincentelli, Maurizio Peri, and Saverio Pezzini. 2003. Fault-tolerant platforms for automotive safety-critical applications. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. ACM, 170–177.
- [2] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. 2017. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 19–24.
- [3] Wonseok Choi, Dongyeob Shin, Jongsun Park, and Swaroop Ghosh. 2019. Sensitivity based error resilient techniques for energy efficient deep neural network accelerators. In *Proceedings of the 56th Annual Design Automation Conference (DAC)*. ACM/IEEE.
- [4] Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech. 2018. Analyzing and increasing the reliability of convolutional neural networks on GPUs. *IEEE Transactions on Reliability* 68, 2 (2018), 663–677.
- [5] Dan Ernst, Shidhartha Das, Seokwoo Lee, David Blaauw, Todd Austin, Trevor Mudge, Nam Sung Kim, and Krisztian Flautner. 2004. Razor: Circuit-level correction of timing errors for low-power operation. *IEEE Micro* 24, 6 (2004), 10–20.
- [6] Richard W Hamming. 1950. Error detecting and error correcting codes. *The Bell System Technical Journal* 29, 2 (1950), 147–160.
- [7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

- [8] Zhezhi He, Jie Lin, Rickard Ewetz, Jiann-Shiun Yuan, and Deliang Fan. 2019. Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping. In *Proceedings of the 56th Annual Design Automation Conference (DAC)*. ACM/IEEE.
- [9] Kuang-Hua Huang and Jacob A. Abraham. 1984. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.* C-33, 6 (1984), 518–528.
- [10] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems*. 4107–4115.
- [11] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).
- [12] Norman P Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. ACM/IEEE.
- [13] Michael Klachko, Mohammad Reza Mahmoodi, and Dmitri Strukov. 2019. Improving noise tolerance of mixed-signal neural networks. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE.
- [14] Israel Koren and C Mani Krishna. 2010. *Fault-tolerant systems*. Elsevier.
- [15] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.
- [16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
- [17] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM.
- [18] Chenchen Liu, Miao Hu, John Paul Strachan, and Hai Li. 2017. Rescuing memristor-based neuromorphic design with high defects. In *Proceedings of the 54th Annual Design Automation Conference (DAC)*. ACM/EDAC/IEEE.
- [19] Tao Liu, Wujie Wen, Lei Jiang, Yanzhi Wang, Chengmo Yang, and Gang Quan. 2019. A fault-tolerant neural network architecture. In *Proceedings of the 56th Annual Design Automation Conference (DAC)*. ACM/IEEE.
- [20] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. 2019. Design of reliable DNN accelerator with un-reliable ReRAM. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1769–1774.
- [21] Mohamed A Neggaz, Ihsen Alouani, Pablo R Lorenzo, and Smail Niar. 2018. A reliability study on CNNs for critical embedded systems. In *36th International Conference on Computer Design (ICCD)*. IEEE, 476–479.
- [22] Elbruz Ozen and Alex Orailoglu. 2019. Sanity-Check: Boosting the reliability of safety-critical deep neural network applications. In *28th Asian Test Symposium (ATS)*. IEEE, 7–12.
- [23] Elbruz Ozen and Alex Orailoglu. 2020. Concurrent Monitoring of Operational Health in Neural Networks Through Balanced Output Partitions. In *25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 169–174.
- [24] Adam Paszke et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8026–8037.
- [25] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference (DAC)*. ACM/ESDA/IEEE.
- [26] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*. ACM/IEEE, 267–278.
- [27] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 979–984.
- [28] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Efficient on-line error detection and mitigation for deep neural network accelerators. In *International Conference on Computer Safety, Reliability, and Security*. Springer, 205–219.
- [29] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmailzadeh. 2016. From high-level deep neural models to FPGAs. In *49th International Symposium on Microarchitecture (MICRO)*. IEEE/ACM.
- [30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
- [31] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* 32 (2012), 323–332.
- [32] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.
- [33] Jeff Zhang, Kartheek Rangineni, Zahra Ghodsi, and Siddharth Garg. 2018. ThUnderVolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In *Proceedings of the 55th Annual Design Automation Conference (DAC)*. ACM/ESDA/IEEE.
- [34] Jeff Jun Zhang, Tianyu Gu, Kanad Basu, and Siddharth Garg. 2018. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *36th VLSI Test Symposium (VTS)*. IEEE.