The Impact of Faults on DNNs: A Case Study

1st Elaheh Malekzadeh
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
elahehma@kth.se

2nd Nezam Rohbani
School of Computer Science
Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
rohbani@ipm.ir

3rd Zhonghai Lu
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
zhonghai@kth.se

4th Masoumeh Ebrahimi
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
mebr@kth.se

***Abstract*—Deep neural networks (DNNs) are showing superior advantages in different domains and are opening their path into critical applications where reliability is the main concern. DNNs can be executed in different hardware platforms, including general-purpose processors which usually operate under floating-point (FP) numbering systems. Considering the small range of weights in DNNs stored in the FP format, some bits remain constant as 0 or 1 for all weights. On the other hand, a single event upset may flip a bit, increasing or decreasing the value of a weight. In this paper, we analyze the effect of bit flips in a sample network of LeNet5, and show the sensitivity of convolution layers to faults and the vulnerability of DNNs to a single fault in a specific bit position. This is while the network is inherently robust against bit flips in the other bit positions. We then show that the choice of activation functions and pooling techniques could alleviate the negative effects of faults to a large extend.**

I. Introduction

THE notable benefits of Deep Neural Networks (DNNs) have led to the advancement in many real-world applications, such as speech recognition and image classification. This has resulted into the recent growing popularity on developing advanced platforms for DNN computation.

The most common hardware platforms to execute DNN operations are CPU, GPU, FPGA, and ASIC. These platforms may be used for executing different DNN architectures and models (e.g., AlexNet, VGGNet, and LeNet) with varying sizes and complexities. Since the accuracy and supported range of floating-point operations are enough for many applications, the general-purpose platforms, like GPUs, operate under floating-point (FP) numbering systems, e.g., single-precision (32-bit). This is while DNN accelerators support fixed-point numbering systems as well. The reason is that the range of values in neural networks can fit well in the fixed-point format. Thereby, DNN computations can be accelerated with a low power consumption in comparison with FP operations. Despite the existence of DNN accelerators, still many DNN applications are executed on general-purpose processors.

978-1-6654-1609-2/21/$31.00 ©2021 IEEE

DNNs are highly resilient against noise and faults. However, the computation platforms are subject to faults, which may occur on hardware, and that is the interest of this paper. Hardware faults can arise from defects in hardware components or from factors that are external to the system. Two major categories of hardware faults are called *permanent* and *transient* faults. Permanent faults are caused by persistent physical flaws in the hardware, such as a short circuit or a grounded wire, caused by process variation or aging. Transient faults, the most common fault model in memory systems, are generally caused by external radiation, electromagnetic interference, and electrostatic discharge [1], [2].

A common way to model faults in digital circuits is using the stuck-at fault model. In this model, the output of a gate is assumed to *stuck* at logical 1, 0, or even an unknown state X. Another common fault model is the random bit flip. Random bit flip model can be used to model single event upset (SEU), multiple bit upset (MBU), and other transient faults that occur in storage elements [3]. The focus of this paper is on bit flip fault models.

In this paper, we investigate a case where a sample network of LeNet5 with a limited range of weight values runs on a platform with a 32-bit FP numbering system. Accordingly, the weights and biases should be stored in the 32-bit FP format. It is worth mentioning that the FP format is supported in almost all of the mainstream hardware platforms. We evaluate the effect of a bit flip in different bit positions of these 32 bits. Our evaluations show that a single bit flip fault may lead to a significant accuracy drop. Utilizing double-precision (64-bit) or half-precision (16-bit) numbering formats does not reduce the error rate. Our findings can be well utilized to apply efficient fault-tolerant methods and adapt techniques to defend adversarial bit flip attacks.

II. Related work

In [3], Torres-Huitzil and Girau review articles that address the fault tolerance of neural networks mostly on a network architecture level using passive fault tolerance methods. In passive fault tolerance techniques, the system uses redundancy and fault masking methods to tolerate faults and does not react to faults in any special *active* way. Some of the methods include pruning by measuring the sensitivity of neurons to faults [4], replicating critical neurons and adding redundancy [5], and fault injection during training to decrease the sensitivity to faults [6].

Dias and Antunes in [5] consider stuck-at fault models and by identifying the importance of elements in a network, they decide to introduce spatial redundancy by duplicating inputs, biases, weights, or whole neurons. This method, however, addresses fault tolerance on a network-level and comes with a trade-off between resiliency and increased hardware resource usage.

Apart from the research on inherent fault tolerance of neural network architectures, studies have been carried out that address the underlying hardware faults and their effect on neural networks [7]–[9]. For this aim, in a series of fault injection experiment, Neggaz et al. [7] studies the layer-wise performance of a CNN network called LeNet [10] after injecting SEU in the memory containing network parameters. Their results show that fully connected layers in LeNet are far more destructive to the accuracy than the convolutional layers. They explain their findings by the fact that errors in fully connected layers propagate directly to the output, while convolutional layers are followed by max-pooling layers that can potentially mask abnormal data values. In [8], Beyer et al. introduces a fault injection framework for networks modeled with Tensorflow. In this framework, faults are injected after the addition, multiplication, or subtraction operations in the model graph. They perform a case study on a CNN network specialized for traffic sign recognition and showed that faults injected in the addition results have the greatest impact on accuracy, lowering it by 60% in extreme cases. Bosio et al. in [9] proposes a fault injection platform targeting Yolo and LeNet networks. Their study is focused on autonomous driving safety standards. The platform can measure how severely injected faults can affect layers in the network and classify if those faults would lead to failure.

The inherent robustness of different networks are evaluated in [11]–[17]. The effect of Bit Flip Attacks (BFAs), injected using row-hammer effect, laser beam, or software-based fault injection, are utilized for this aim. In general, these works suggest that larger DNNs are more robust against BFAs, and different activation functions are effective on preserving accuracy of the network due to the injected fault.

Characterizing the susceptibility of network to each bits' fault in the weights of the network is presented in [18]–[26]. These works show that bit flip in different bits of each weight have different effect on the network output accuracy when considering a fixed-point or floating-point numbering system. The accuracy of the network is more degraded due to bit flip in the most significant bits in the fixed-point and the exponent bits in the floating-point numbers.

In this paper, we rely on the previous works and observations which validate our work on larger DNNs. However, by studying a small network, LeNet5, we explain in detail why some specific bits are more susceptible to faults. Our characterization of susceptibility of LeNet5 to faults can be well generalized to larger networks. Our evaluation can be utilized to design low-overhead and effective fault-tolerant techniques. Furthermore, our results specify the bit positions where adversarial attacks can cause a severe network accuracy drop. In addition, by evaluating the weights' bit positions in LeNet5, as a case study, a comprehensive explanation of bits susceptibility is presented in this work. Finally, activation functions and pooling methods are investigated as potential techniques to mask faults.

III. Preliminary

A. Floating-point Numbering System

Digital signals representing real and integer values in computer systems can be represented and stored in a fixed-point or floating-point format. The fixed-point format has a fixed number of digits after the decimal point. On the other hand, in the floating-point format, a number is expressed using four terms: a *sign*, a *mantissa*, a *base*, and an *exponent* as shown in Equation 1.

$$\pm mantissa \times base^{exponent} \tag{1}$$

To represent a floating-point number in a computer system, an IEEE754 *single precision* floating-point format (FP32) is employed, composed of a sign, a mantissa, and an exponent, as shown in Fig. 1.

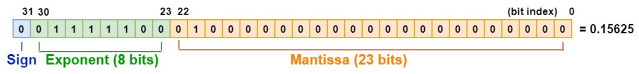


Fig. 1: IEEE754 Single-precision floating-point format

In this standard, sign, mantissa and exponent have 1, 23 and 8 bits, respectively. However, due to the *leading bit convention* rule, there is an implicit 1 as the most significant bit of mantissa which serves as the integer part of the number, making the mantissa 24 bits in total, with 23 bits of fractional part that is stored in this format. Moreover, the exponent part is encoded with *excess-127* coding, which simply adds 127 to the exponent before storing it.

B. LeNet5

The DNN studied in this work is LeNet5 [10], shown in Fig. 2, which is composed of an input layer, two convolution layers each followed by a pooling layer, three fully connected layers, and an output layer. The activation functions are rectified linear function (ReLU) except the last layer which is the Sigmoid function.

In Fig. 3, we draw the histogram of network weights. The histogram shows that the network parameters are distributed quite symmetrical and narrowly in the range $[-0.6, +0.6]$, with most of them being between $[-0.1, +0.1]$. Table I shows the number of trainable weights in each layer of LeNet5 that can be subject to faults.

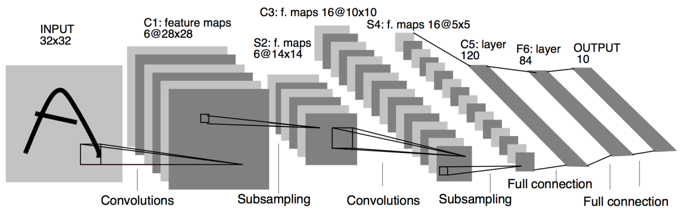


Fig. 2: LeNet5 architecture [10]

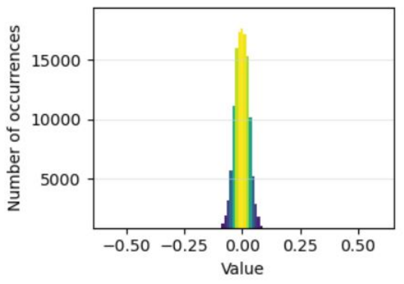


Fig. 3: Histogram of trained weights in LeNet5

IV. Bit Flip in a Floating-point Numbering System

A. The Fault Injection Framework

The fault injection framework takes a specific layer, and by using a random number generator with a uniform distribution, it chooses a parameter within the layer. Then, taking the chosen parameter, again with a random number generator with a uniform distribution, chooses one of 32 bits of it. It then flips the logic state of that bit, i.e., if it was a '1', turns it to a '0' and vice versa. The last step is writing the *faulty* parameter back to the storage so as to make the network *seem* faulty.

The procedure is repeated as many times as specified by the number of faults. Afterwards, the framework performs accuracy measurement on the test dataset using the now faulty network. The framework records this accuracy, then performs all of the steps above as per number of iterations, recording the resulting accuracy for each one. Lastly, it averages all of the measured accuracy values and produces desired output graphs for evaluation and statistical analysis. The pseudo-code is shown in Algorithm 1.

The hardware faults or SEUs are modeled as bit flips in the data that is stored in the memory. Data in this context are the parameters of the DNN, i.e., weights of each layer. The fault is injected by flipping the selected bit of the randomly chosen parameter. It is worth mentioning that all of the other fault models which lead to multiple bit flips in memory cells, e.g., MBU, can be modeled with this procedure. However, to exactly characterize the susceptibility of each bit in the network weights, we only considered the SEU.

Algorithm 1: The Fault Injection Procedure

```
Function Fault_Evaluation:
    Take a copy of the network;
    Take a layer of the network;
    for i in range(Iterations) do
        for j in range(Faults) do
            Randomly pick a weight in the layer;
            if The weight has not been picked yet then
                Transform the FP32 weight to binary;
                Randomly pick a bit position between 0-31;
                Flip the bit;
                Transform the weight back to FP32 and store it;
        Evaluate the faulty network with the test dataset;
        Calculate and store accuracy;
    return Average accuracy over Iterations;
```

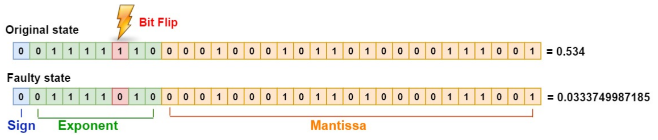


Fig. 4: Bit flip on a parameter and the change of the value

An example of fault injection procedure is shown in Fig. 4, where the 25th bit position of a hypothetical network parameter in FP32 format is flipped, resulting in a change of the value in the stored parameter. This example shows how a random fault causes the number to shrink down by a factor of 16.

B. Impact of Faults on a FP Numbering System

1) Faults in the bits of Exponent: Fig. 5 plots the histogram of exponent values in binary. As it can be seen, the majority of the weights in LeNet5 have the binary exponent value of 01111001, where the three most significant bits are identical in all of them. Let us now have a closer look at this exponent value 01111001, which corresponds to the decimal number 121. Since in IEEE-754, exponent is encoded with excess-127, then 121 is decoded as -6. Given this value, the mantissa is multiplied by the factor of 2^{-6}.

Now, if we inject a fault in the most significant bit of the exponent (i.e., 8th bit of the exponent or the 30th bit position in Fig. 1), the resulting value would be $11111001 = 249$. After deducting 127, the corresponding exponent would be 2^{122}, making the weight value extremely large. However, injecting a fault in the 7th bit position of the exponent would make the exponent equal to $00111001 = 57$ which corresponds to the exponent of 2^{-70}, shrinking the weight value to near-zero. This implies that the weight is removed from the calculations, and somewhat pruned. In short, flipping '0' to '1' makes the exponent larger while flipping from '1' to '0' makes it smaller.

This analysis shows that the bit flip impact is significant only in the most significant bit of the exponent, which may grow a weight by a factor of 2^{122}, and thus suggesting a single point of failure. This is far beyond the range of other parameters available in the network and can quickly cause saturation, overflow, or other problems in computation units. If no masking, limiting, or any other measures are taken, this large faulty weight can propagate in the network and disrupt all calculations it is involved in. To confirm this finding, we flip all of 32 bits of a weight one by one and in isolation from each other, and evaluate the accuracy each time. The result in Fig. 6 indicates a significant drop in accuracy (from 99% to about 10%) when a fault is injected in the 30th bit position.

A similar concept applies to double-precision (FP64) and half-precision (FP16) floating point numbering systems. The fixed-point numbering system would alleviate the impact of

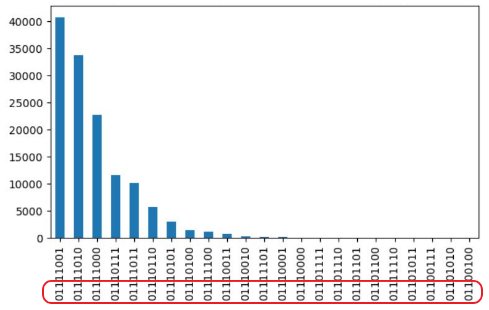


Fig. 5: Histogram of weights' exponent values in LeNet5

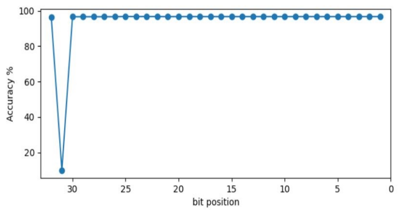


Fig. 6: Accuracy vs. bit position of the fault

bit flip faults. However, fixed-point operation is not commonly supported by general-purpose CPUs, GPUs, and compilers.

2) Faults in the bits of Sign and Mantissa: Faults in the bits of sign and mantissa have a limited impact on the range of values. A bit flip on the sign bit turns a positive value to a negative value and vice versa while the range of values remains in the common range of $[-0.1, +0.1]$ (Fig. 3). Given that the bits of mantissa contribute in the fraction part of a number, in the extreme case, a bit flip in the most significant bit of the mantissa changes a number's value by -0.5 or $+0.5$. This implies that the impact of a bit flip fault on the sign bit is larger than that of mantissa, while in both cases, the impact is way smaller than that of the exponent bit.

C. Masking Faults

We investigate two solutions to mask faults and prevent the faulty weights with large values from propagating to the subsequent layers. One solution would be limiting the values by the choice of the non-linear activation function and the other solution through a right choice of the pooling layer.

1) Relu vs. Sigmoid: In ReLU, the function grows as the input increases and there is no constraint on the output of a neuron. The Sigmoid function, on the other hand, has a bounded range between $(0, 1)$, thereby we expect that the Sigmoid function could prevent the propagation of extremely large weights to the output.

Regarding the hardware complexity of these two activation functions, ReLU is very efficient in terms of computations compared to Sigmoid. A simple unit that compares the input to zero, which can be made very small with logic elements, is sufficient to produce ReLU on hardware. Oppositely, the Sigmoid function has an exponential term as well as a fractional term. Sigmoid in its simplest form can be implemented as a piece-wise linear approximation or a look-up table that contains samples of the original function. Depending on the resources available in the hardware, more precise and complex implementations of Sigmoid can be implemented using hardware or software.

2) Average vs. Max pooling: In LeNet5, each convolution layer is followed by a pooling layer to reduce the size of the feature map. There are different types of poolings as min pooling, max pooling, and average pooling. Max pooling is the one used in LeNet5, and the most popular one in the image classification tasks. While max pooling returns the maximum value in the portion of the feature map, average pooling returns the average of all values in the given window. As bit flip faults, like SEUs, tend to shrink or enlarge values, we expect that the use of average pooling may smooth out this effect compared to min or max pooling which likely selects the faulty values.

V. Result and Discussion

LeNet5 was trained on the MNIST dataset [27] using the stochastic gradient descent optimizer with a learning rate of 0.001 and a batch size of 4. Pytorch [28] framework is used for building LeNet5, training, testing and loading the MNIST dataset. The baseline classification accuracy of 99.02% on the test data set was achieved after training for 10 epochs.

A. Sensitivity of Different Layers to Faults

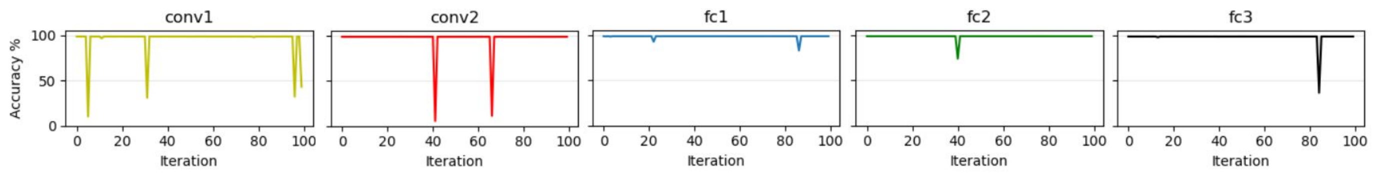
In the experiments, illustrated in Fig. 7, we look at the faults in each layer of the network. The fault percentages are presented in Table I for 1, 10 and 20 bit flips in each of the layers. It is worth mentioning that the location of particle strike and circuit topology is effective on the number of bit flips, and usually the number of bit flips is less than 4 bit flips [29]. On the other hand, some fault models like stuck-at faults due to aging or process variation can lead to larger number of concurrent faults.

TABLE I: Number of parameters and fault percentage in each layer of LeNet5.

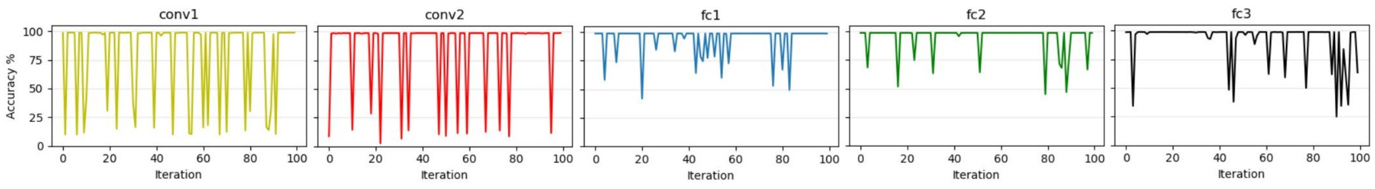
Layer	Number of weights	Fault Percentage (%)		
		1 fault	10 faults	20 faults
Conv1	500	0.2	2	4
Conv2	25000	0.004	0.04	0.08
FC1	96000	0.0010	0.0104	0.0208
FC2	10080	0.0099	0.0992	0.1984
FC3	840	0.1190	1.1904	2.3809

Faults are injected randomly, and the impact of bit flips is reported for each of the 100 iterations. Obviously, injecting more faults (Fig. 7a to Fig. 7c) leads to the increased cases of accuracy drop.

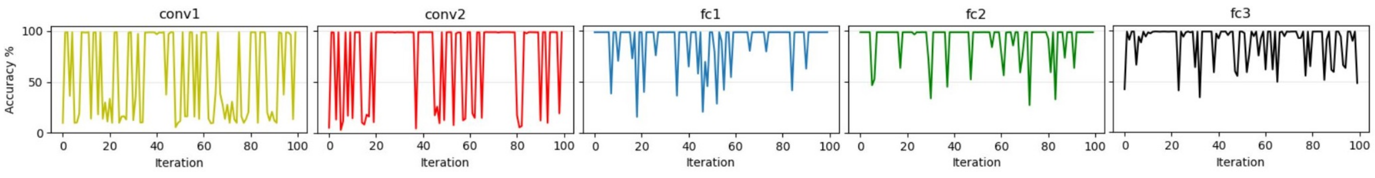
Another insight from Fig. 7 is that convolution layers are more prone to faults than fully connected layers even if their fault percentages are lower (see Table I). One possible reason is that if the layer is among the first layers in the sequence of layers, the propagation of a very large valued faulty weight creates a chain of subsequent large intermediate results that spreads much more into the network compared to the case when the layer is among the last layers. These experiments have been repeated for up to 2000 iterations. The scatter plots for the Conv1 and Fc3 layers are illustrated in Fig. 8a and 8b.



(a) 1 bit flip

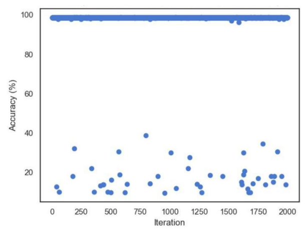


(b) 10 bit flips

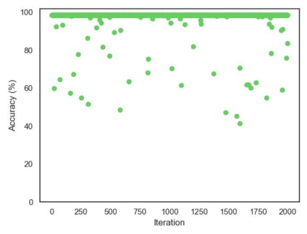


(c) 20 bit flips

Fig. 7: The impact of 1, 10, and 20 bit flips on the accuracy when applied on each layer of LeNet5



(a) Conv1



(b) Fc3

Fig. 8: Scatter plot over 2000 iterations for 1 bit flip.

B. *Sensitivity of Bit Positions to Faults*

Our analysis show that the accuracy is almost unaffected by flipping the bits of sign and mantissa in a single weight, regardless of the layer that the chosen weight belongs to. However, as shown in Fig. 9, faults in the bits of exponent can be far more effective in the absolute value of a floating-point number. The experiments were repeated with 10 and 20 bit flips and the same pattern was observed.

C. *Sigmoid vs. ReLU*

Using the insights from Section V-A, this experiment focuses only on the two most vulnerable layers: Conv1 and Conv2. Additionally, only the exponent bit field is subject to fault injection in this experiment. Using Sigmoid instead of ReLU without any further hyper-parameter tuning or modifications of the network, decreased the network accuracy by 1%. The resulting network was injected with 20 bit flips in the exponent bit field of 20 distinct weights. Accuracy of the network was evaluated after the fault injection and the whole process was repeated for 100 times. Fig. 10 shows the result of this experiment, confirming the effectiveness of the Sigmoid function in masking faults, as compared to ReLU in Fig. 7c.

D. *Average Pooling vs. Max Pooling*

In this experiment, the original LeNet5 with max pooling layers was modified to use average pooling layers as a measure to lower the effect of substantially large valued faulty weights by averaging over a window of numbers after each layer in the network. After this change, 100 iterations of injecting 20 bit flips in the exponent bit fields of network parameters were performed and the accuracy of each iteration was evaluated, shown in Fig. 11. The number of accuracy dips due to the fatal fault occurrence decreased by using average pooling. However, the worst case fault still managed to seep through the next layers and demolish the accuracy in a few iterations. Based on the results, even though average pooling is more capable of preventing the propagation of faults compared to max pooling, it is still not as effective as a limited-range activation function.

VI. Conclusion

DNNs could enhance accuracy in critical applications but ensuring reliability remains an issue. Through a case study, we looked into the impact of bit flip faults in DNNs when executing on the general-purpose processors operated under the floating-point numbering systems. We analyzed and explained how a single bit flip in the most significant bit of the exponent may result in dropping the accuracy from 99% to about 10%. This indicates a single point of failure which demands extra measures and protections. We then examined different activation functions and pooling techniques to prevent propagating the faults to the subsequent layers of the network. We believe these findings can be utilized in the design of efficient fault-tolerant techniques against adversarial bit-flip attacks.

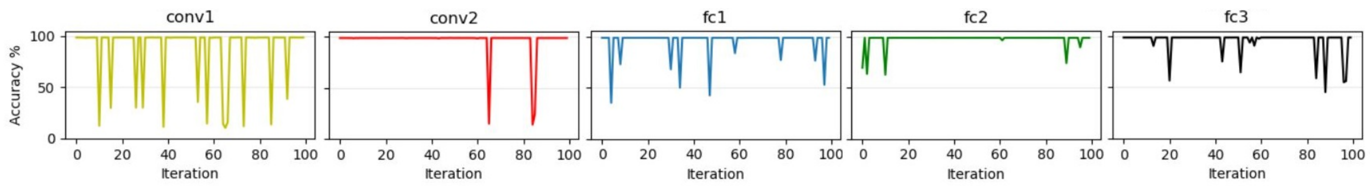
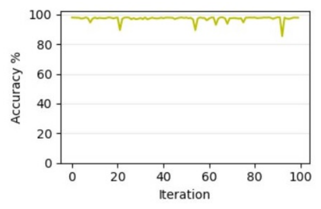
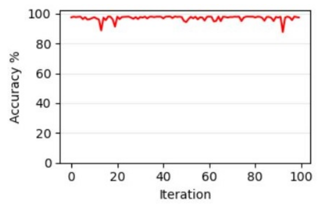


Fig. 9: The impact of 1 bit flip on the exponent

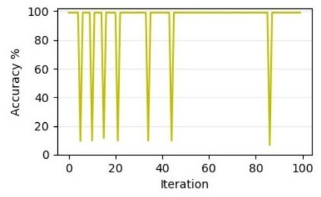


(a) Conv1

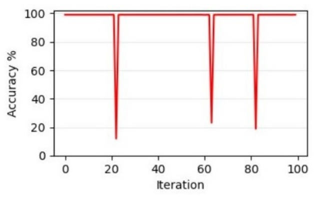


(b) Conv2

Fig. 10: Effect of 20 bit flips on accuracy using Sigmoid



(a) Conv1



(b) Conv2

Fig. 11: Effect of 20 bit flips on accuracy with Average Pooling

Acknowledgment

The research was supported in part by VINNOVA (Sweden's Innovation Agency) through the Trust-E project of the Eureka PENTA and EURIPIDES programmes. It was also supported by VR (Swedish Research Council) through the CSR project.

References

[1] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "Safer: Stuck-at-fault error recovery for memories," in *43rd IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 115–124.

[2] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 297–310, 2015.

[3] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017.

[4] C. . Chiu, K. Mehrotra, C. K. Mohan, and S. Ranka, "Robustness of feedforward neural networks," in *IEEE International Conference on Neural Networks*, 1993, pp. 783–788 vol.2.

[5] F. M. Dias and A. Antunes, "Fault tolerance improvement through architecture change in artificial neural networks," in *International Symposium on Intelligence Computation and Applications*, 2008, pp. 248–257.

[6] C. H. Sequin and R. D. Clay, "Fault tolerance in artificial neural networks," in *1990 IJCNN International Joint Conference on Neural Networks*, 1990, pp. 703–708 vol.1.

[7] M. A. Neggaz, I. Alouani, P. R. Lorenzo, and S. Niar, "A reliability study on cnns for critical embedded systems," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 476–479.

[8] M. Beyer, A. Morozov, K. Ding, S. Ding, and K. Janschek, "Quantification of the impact of random hardware faults on safety-critical ai applications: Cnn-based traffic sign recognition case study," in *IEEE International Symposium on ISSREW*, 2019, pp. 118–119.

[9] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network," in *IEEE LATS*, 2019, pp. 1–6.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[12] L. Liu, W. Wei, K.-H. Chow, M. Loper, E. Gursoy, S. Truex, and Y. Wu, "Deep neural network ensembles against deception: Ensemble diversity, accuracy and robustness," in *IEEE MASS*. IEEE, 2019, pp. 274–282.

[13] C. D. Schuman, J. P. Mitchell, J. T. Johnston, M. Parsa, B. Kay, P. Date, and R. M. Patton, "Resilience and robustness of spiking neural networks for neuromorphic systems," in *Proc. IJCNN*. IEEE, 2020, pp. 1–10.

[14] J. Breier, X. Hou, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Practical fault attack on deep neural networks," in *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2204–2206.

[15] A. P. Arechiga and A. J. Michaels, "The effect of weight errors on neural networks," in *Proc. IEEE CCWC*. IEEE, 2018, pp. 190–196.

[16] N. Khoshavi, S. Sargolzaei, Y. Bi, and A. Roohi, "Entropy-based modeling for estimating adversarial bit-flip attack impact on binarized neural network," in *Proc. ASP-DAC*. IEEE, 2021, pp. 493–498.

[17] A. S. Rakin, L. Yang, J. Li, F. Yao, C. Chakrabarti, Y. Cao, J.-s. Seo, and D. Fan, "Ra-bnn: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy," *arXiv preprint arXiv:2103.13813*, 2021.

[18] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *Security Symposium ({USENIX} Security 20)*, 2020, pp. 1463–1480.

[19] Z. Yan, Y. Shi, W. Liao, M. Hashimoto, X. Zhou, and C. Zhuo, "When single event upset meets deep neural networks: Observations, explorations, and remedies," in *Proc. ASP-DAC*, 2020, pp. 163–168.

[20] W. Choi, D. Shin, J. Park, and S. Ghosh, "Sensitivity based error resilient techniques for energy efficient deep neural network accelerators," in *Proc. of DAC*, 2019, pp. 1–6.

[21] J. Li, A. S. Rakin, Y. Xiong, L. Chang, Z. He, D. Fan, and C. Chakrabarti, "Defending bit-flip attack through dnn weight reconstruction," in *Proc. of DAC*. IEEE, 2020, pp. 1–6.

[22] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraș, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *Security Symposium ({USENIX} Security 19)*, 2019, pp. 497–514.

[23] M. A. Hanif, F. Khalid, R. V. W. Putra, S. Rehman, and M. Shafique, "Robust machine learning systems: Reliability and security for deep neural networks," in *Proc. IEEE IOLTS*. IEEE, 2018, pp. 257–260.

[24] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1211–1220.

[25] J.-S. Kim and J.-S. Yang, "Dris-3: Deep neural network reliability improvement scheme in 3d die-stacked memory based on fault analysis," in *Proc. of DAC*. IEEE, 2019, pp. 1–6.

[26] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, "T-bfa: Targeted bit-flip adversarial weight attack," *arXiv preprint arXiv:2007.12336*, 2020.

[27] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST Database of Handwritten Digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[28] A. Paszke and et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[29] S. Kumar, M. Cho, L. Everson, Q. Tang, P. Meinerzhagen, A. Malavasi, D. Lake, C. Tokunaga, M. Khellah, J. Tschanz *et al.*, "Analysis of neutron-induced multibit-upset clusters in a 14-nm flip-flop array," *IEEE Transactions on Nuclear Science*, vol. 66, no. 6, pp. 918–925, 2019.