Evaluating Resilience of Deep Learning Models

Evaluando la Resiliencia de Modelos de Deep Learning

Elvis Rojas[1], Bogdan Nicolae[2], Esteban Meneses[3]

Rojas, E; Nicolae, B; Meneses, E. Evaluating Resilience of Deep Learning Models. *Tecnología en Marcha*. Edición especial 2020. 6th Latin America High Performance Computing Conference (CARLA). Pág 25-30.

https://doi.org/10.18845/tm.v33i5.5071

1 Associate Professor, School of Informatics, National University of Costa Rica. Dr-Ing Student, Doctor of Engineering program, Costa Rica Institute of Technology. Costa Rica. E-mail: erojas@una.ac.cr.
https://orcid.org/0000-0002-4238-0908

2 Computer Scientist, Mathematics and Computer Science, Argonne National Laboratory, United States. E-mail: bogdan.nicolae@acm.org.
https://orcid.org/0000-0002-0661-7509

3 Director, Advanced Computing Collaboratory, Costa Rica National High Technology Center. Associate Professor, School of Computing, Costa Rica Institute of Technology. Costa Rica. E-mail: emeneses@cenat.ac.cr.
https://orcid.org/0000-0002-4307-6000

Keywords

Resilience; fault tolerance; deep learning; fault injection.

Abstract

Deep learning applications have become a valuable tool to solve complex problems in many critical areas. It is important to provide reliability on the outputs of those applications, even if failures occur during execution. In this paper, we present a reliability evaluation of three deep learning models. We use an ImageNet dataset and a homebrew fault injector to make all the tests. The results show there is a difference in failure sensitivity among the models. Also, there are models that despite an increase in the failure rate can keep the resulting error values low.

Palabras clave

Resiliencia; tolerancia a fallas; deep learning; inyección de fallos.

Resumen

Los modelos de Aprendizaje Profundo se han convertido en una valiosa herramienta para resolver problemas complejos en muchas áreas críticas. Es importante proveer confiabilidad en las salidas de la ejecución de estos modelos, aún si se producen fallos durante la ejecución. En este artículo presentamos la evaluación de la confiabilidad de tres modelos de aprendizaje profundo. Usamos un conjunto de datos de ImageNet y desarrollamos un inyector de fallos para realizar las pruebas. Los resultados muestran que entre los modelos hay una diferencia en la sensibilidad a los fallos. Además, hay modelos que a pesar del incremento en la tasa de fallos pueden mantener bajos los valores de error.

Introduction

Machine learning (ML) has become a growing research area and it has applications from financial services to image recognition. Much of this growth has been fueled by the rise of deep neural networks (DNN). DNNs automatically learn based on a training dataset to recognize patterns to classify objects like images or audio signals. With the increase of interest in DNNs many deep learning (DL) frameworks have been developed (TensorFlow, Pytorch, MXNet, DeepLearning4j and Keras) to facilitate the development and implementation of DNNs. It is crucial to provide reliability to the outputs of DNNs to ensure the accuracy and correctness of the results, even if there are failures in the execution of the model. Therefore, there is a growing need in understanding the resilience of DNN models to analyze how they are affected by failures. The goals of this evaluation are: i) to analyze the behavior of real DNN models in the presence of failures and ii) provide results that can be used by designers or developers to improve the fault tolerance mechanisms of their models.

Methodology

In this paper, we present the evaluation of the resilience of three pretrained DNN models. Two of them are convolutional neural networks (CNN) and one is a residual neural network (RNN). These models were implemented in Python using the Pytorch framework, since Pytorch tensors can be used and manipulated like NumPy arrays and therefore more amenable to our research purposes. Also, Pytorch provides the possibility of implementing pretrained models with the ImageNet dataset.

Failure Injection

To test the resilience of DNN models, we developed a program capable to inject failures into a DNN model. There are many applications or APIs to perform fault injection (FI) [1][2][3] in programs, but these are not specifically designed to perform the FI in DNN models. Nevertheless, there are other studies that have developed and implemented FI mechanisms to understand the relationship between fault rate and model accuracy, and to test the resilience of DNN models [4] [5][6][7]. Our work differs from these papers due to the fact that we focus on the development of experiments with different failure rates and types of perturbations using our failure injection program. Also, we perform the evaluation using a modern and popular deep learning framework, a different image dataset and different deep learning models to those presented in the literature.

The fault injector was developed to inject failures into the weights of the layers of a model. We get the layers of the DNN to manipulate later the tensors that they contain. To determine where to perform the injection, the program randomly selects the layers and the weights. The fault injector uses three parameters to perform the injection: i) model in which the failures are injected, ii) failure rate, is the factor to get the total number of failures to be injected, iii) perturbation type, which is the type of failure to be injected.

The fault injector determines the number of layers of each model. For each layer there was an extraction of a tensor that contains all the pretrained weights. Also, it is necessary to extract the shape of the layer to determine the size of its components (number of output channels and the number of filters with its weight and height). With the size, we can determine in which item to perform the injection and we can calculate the total amount of weights of a model to apply the failure rate factor.

Parameters of the experiment

We implemented the failure injection in three DNN models: VGG19, RESNET50, and InceptionV3. These models were pretrained with the ImageNet dataset. We used two factors in the experiment to change the behavior of the fault injector: i) The failure rate factor *(frate*). We used three different *frates* 0.00001, 0.00003 and 0.00005 to determine the number of failures to be injected based on the total weights of a model. These frates were selected based on a previous experiment, in which we tested the sensibility of a model with different failure injections. ii) The perturbation factor, which may be either all values set to zero or all values set to random values.

Experiment execution

ImageNet dataset was used with 1,000 different images to perform each experiment and replicas. We executed 21 experiments, 3 without failures (1 experiment per model) and 18 experiments with failure injection in a different factors combination (3 models times 3 frates times 2 perturbations). For each of the 18 experiments, we performed 10 replicas.

Results Analysis

To analyze the results of the experiments we used descriptive statistics and a bit of analysis of variance (ANOVA). Descriptive statistics help us to determine the dispersion and the symmetry of the results (error data) and to find outliers that could perturb the analysis. The analysis of variance provides us with a statistical test of whether two or more population means are equal to determine differences among the models regarding its failure resilience.

Experimental results

After the experiment execution, we got 183 mean error values (3 from model execution without failure and 180 from replicas). Figure 1 shows the behavior of the three models without failure and with three different frates values. Note that in all cases InceptionV3 has the lowest error value, showing that is the model with the highest classification error on average. Also, we can see that the models were susceptible to the increment of the frate and only the InceptionV3 model kept the error stable with frates 0.00003 and 0.00005. Note, that the InceptionV3 model is not affected in the same way by the frate as with the other two models. Also, we can see that the VGG19 and ResNet50 models are very similar. These two models have a similar behavior regarding the increase in failures.

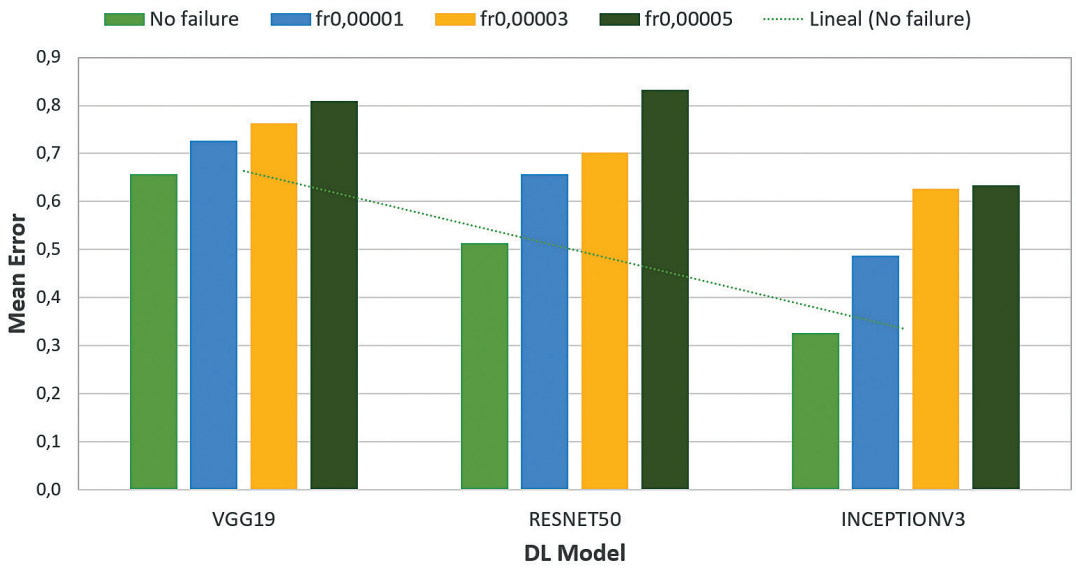

Figure 1. Model sensitivity to failures

We also analyzed the mean error of the replicas per each model and frate. Figure 2 shows that the VGG19 model has the lowest dispersion. The mean error values of VGG19 in all replicas were similar, unlike the ReNet50 and InceptionV3 that show high dispersion, especially with the frate 0.0003. The low dispersion in VGG19 shows that the mean error results of each replica did not differ significantly. Also, note that in with frate=0000.1 there is a negative skew in the interquartile range (IQR) and the minimum and maximum are similar, showing that the failure injection with a frate=0.00001 did not significantly impact in the classification process of all models. The InceptionV3 model shows a positive skew with the frates 0.00003 and 0.00004. Despite the InceptionV3 has the lowest error, it shows that the results with the FI can significantly vary between them.

Figure 3 shows standard deviation ranges that are part of the test for equality of variances. We can see that the frate intervals overlap on each model. That is an indicator that the standard deviation of the results in the three models was not significantly different. Also, we determine that there is equality of variances among the three frates of the ResNet50 and InceptionV3 models. The test for VGG19 determines that there was a statistically significant difference to accept the null hypothesis (all the variances are equal) due that the p-value < 0.05. Based on the former premise, we can conclude that the ResNet50 and Inception Models have a similar behavior through different FI rates.

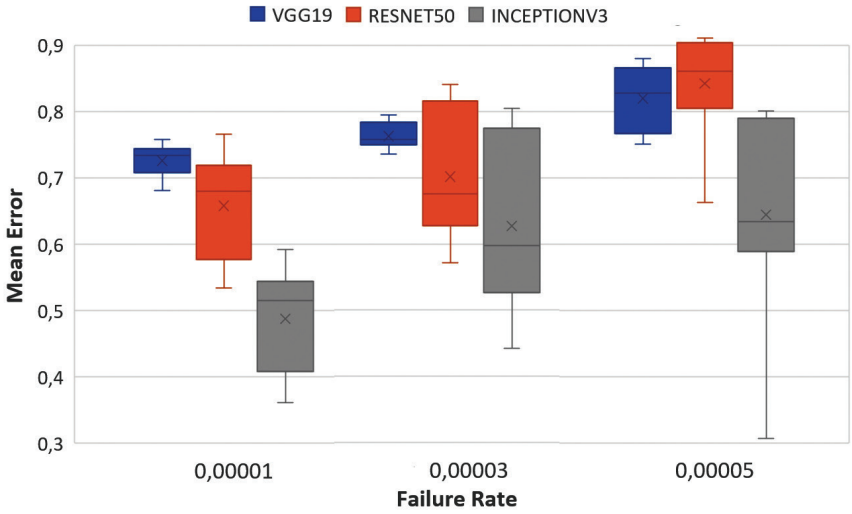

Figure 2. Model reliability according to frates

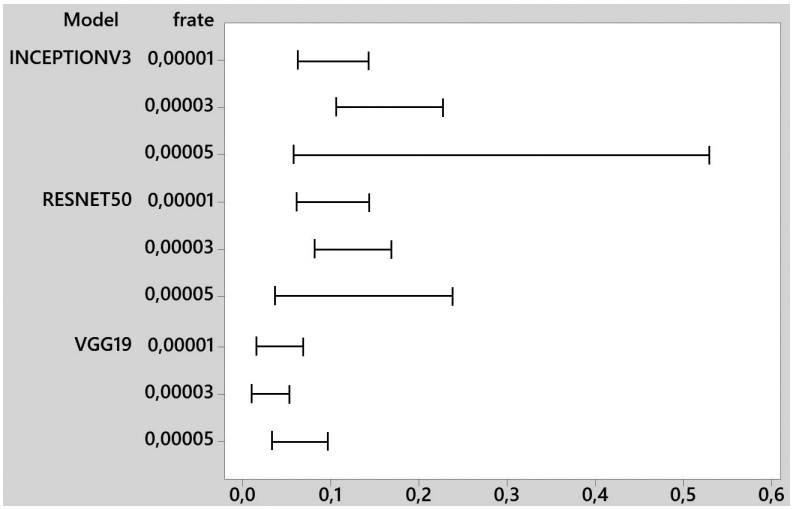

Figure 3. Standard deviation ranges

Final remarks

The experiments provide a useful review of the behavior of three real DNN models under three different failure injection rates. The use of descriptive statistics and the analysis of variance helps us to analyze the results, which can be used to deeply analyze why the deep learning models are affected differently in the presence of failures. Also, the statistical analysis could help other researchers develop methods and techniques to improve the resilience of deep learning models.

We conclude that the models were susceptible to the increase of failures in the classification process and there are models that despite the failures increase can keep the resulting error values low. We plan on extending the current analysis, expanding the statistical analysis with implementing more complex experiments. Also, it is important to analyze other scenarios, increasing the number of models, types of perturbations and the number of replicas.

References

[1] Qining Lu, Mostafa Farahani, Jiesheng Wei, Anna Thomas, and Karthik Pattabiraman. Llfi: an intermediate code-level fault injection tool for hardware faults. In Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on Software Quality, Reliability and Security.

[2]. K. S. Hari, S. Adve, H. Naeimi, and P. Ramachandran, “Relyzer: exploiting application-level fault equivalence to analyze application resiliency to transient faults,”.

ACM SIGARCH Computer Architecture News , vol. 40, p. 123, 04 2012.

[3] U. Schiffel, A. Schmitt, M. Süßkraut and C. Fetzer, “Slice Your Bug: Debugging Error Detection Mechanisms Using Error Injection Slicing,” 2010 European Dependable Computing Conference, Valencia, 2010, pp. 13-22.

[4] TensorFI: A configurable fault injector for TensorFlow Applications”, Guanpeng Li, Karthik Pattabiraman and Nathan DeBardeleben, 8th IEEE International Workshop on Software Certification (WoSoCER), 2018.

[5] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, “Ares: A framework for quantifying the resilience of deep neural networks,” in Proceedings of the 55th Annual Design Automation Conference, ser. DAC ’18. New York, NY, USA: ACM, 2018, pp. 17:1–17:6.

[6] Y. Liu, L. Wei, B. Luo and Q. Xu, “Fault injection attack on deep neural network,” 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, 2017, pp. 131-138.

[7] G. Li, S. K. 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,” in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC ’17. New York, NY, USA: ACM, 2017, pp. 8:1–8:12.