Input Prioritization for Neural Networks

Suat Tarık Demirel

Abstract—**Input prioritization is a technique that can be used to improve the efficiency of testing DNNs. It ranks the test inputs based on their importance or relevance to the task at hand and uses this ranking to prioritize which inputs should be tested first. In this way, it can help to find error-revealing test cases more quickly and with fewer test input requirements. Thus, it can reduce the cost and time required for labeling the test inputs. In this paper, several different input prioritization methods for DNN testing are evaluated. These methods include softmax-based standard deviation and Euclidean distance methods, and top-k neuron-coverage-based methods. The performance of these methods are evaluated in experiments.**

Index Terms—**input prioritization, DNN, softmax, euclidean distance, top-k neuron-coverage**

I. Introduction

Deep neural networks were gained popularity and success in various domains like autonomous vehicles, face and voice recognition, speech processing, medical diagnosis, and software engineering as well in the last few years. Some of these systems have mostly based on DNN and even small misbehavior can cause fatal consequences in these kinds of systems. This brings considerable risks to safe-critical systems that rely on DNNs heavily.

In traditional safety-critical systems, formal verification techniques can be used to verify essential properties of DNN. However, these methods are very hard to scale and apply for the more complex and larger models. That is why testing is standard practice for evaluating DNN systems since it is more practical and easy to apply. However, testing on DNN systems is much more different from traditional software. In traditional software, the business logic of software is constructed by a software developer, so all possible corner case for software is known and test cases can be prepared accordingly. On the other hand, the business logic behind the software is constructed by DNN itself for DNN software. Thus, a huge amount of test input should be used in order to find all possible corner cases that cause an error or misbehavior. Even though, this huge amount of test data can be collected easily, the cost of labeling this data is very high.

In the DNN context, the test cases that cause the system behaves correctly do not give much inside as test cases that cause an error or misbehavior. To overcome this labeling problem input prioritizing can be used. By prioritizing inputs, these errors revealing test cases are found earlier. With that not only the efficiency of DNN testing is improved but also labeling costs is decreased. Furthermore, model accuracy can be assessed with much fewer test cases.

In this paper, several different methods are evaluated with experiments to prioritize test inputs to find failed test cases before successful test cases. Some of these methods have become very successful to prioritize inputs. However, some of the methods could not show a great difference from the random cases even though they get promising results.

II. Related Work

In the literature, researchers have proposed and evaluated various methods for prioritizing the inputs or features of the data based on their importance or relevance to the task at hand. For, instance Byun proposed that [1] prediction confidence of the model can be used as a prioritizing score. They create their own score metric by using the softmax output of the model. Furthermore, they also used bayesian uncertainty and input surprise to create prioritizing metrics. Another example DeepGini [2], also uses confidence to prioritize inputs. Moreover, they leveraged neuron-coverage methods to create neuron-coverage-based prioritizing metrics.

In this work, we also used prediction confidence for prioritizing tasks. We tried to improve confidence-based methods by using different approaches. Results from these methods are compared with Byun's methods to see whether there is any improvement exists or not. Also, some of the neuron coverage metrics from DeepGauge [3] will be used to create a layer-level neuron prioritizing score.

III. Input Prioritization Methods

A. Softmax Output-Based Scores

Softmax a.k.a softargmax is a mathematical function that converts a vector of K real numbers into a vector of a normalized probability distribution. It is usually used in the last layer of a neural network-based classifier in order to examine the possibility of each label and select the highest probability as the prediction value.

The output of the softmax function gives great insight about the model's uncertainty. If the probability vector has one very high probability while the other seems low, this means the model is certain about the outcome of the prediction. On the other hand, if the probability vector has very close values that do not include any relatively very high values, this means the model is not sure about the outcome of the prediction. That is why this could be a misprediction. Thus, an error-revealing case. We used these key ideas to create softmax-based prediciton scores.

1) Standard Deviation of Softmax Output: Standard deviation is a great measure of the dispersion of a dataset. It gives insight about the predation of the dataset. A low standard deviation means that the values of the dataset are close to the mean of the dataset. On the other hand, a high standard deviation means values mean that the values of the dataset are spread out over a wider range. Thus, when we used standard deviation on the softmax output array, a low standard deviation pointed out that the model is certain about the prediction, and a high standard deviation demonstrated that the model is uncertain about the prediction. We take the standard deviation of the softmax output of each prediction in test data and use it directly as a prioritizing score.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$$

Fig. 1: Formula of Standard Deviation

2) l2 Distance-Based Softmax Output Score: In order to evaluate the distribution of the output of softmax with other than standard deviation, we used Euclidean distance from the average for each label. Then, all distance of each label is summed to create a prioritizing score. Similar to the standard deviation, a low score indicates that each output is close to the average, and a high score points out that outputs spread out over a wider range. Thus we use the following formula to create our prioritizing metric.

$$score(x_i) = \sum_{c=1}^{C} p_{i,c} \left| p_{i,c} - avg(C) \right|$$

Fig. 2: Formula of l2 Distance-Based Softmax Output Score

B. Layer-Level Top-k Neuron Prioritizing Score

Layer-level top-k neuron coverage is proposed as a measurement of the number of the most active k neuron for each layer in DeepGauge. The key idea behind this coverage metric is that neurons from the same layer usually have similar roles, and top active neurons can give insight into the essential functionality of the DNN.

We try to create a prioritizing metric based on top-k neuron coverage. In the training phase, top-k neurons are determined for each layer. In the testing phase, the output of these top-k neurons on each layer is summed for each test input. If the summation result is low, that means the test input could be a corner case that can cause an error or misbehavior.

IV. Experiment

All implementations were done in python with the torch library.

A. Model and Dataset

For both training and prioritizing tasks, a popular library MNIST was used. It is an image library that is used for handwritten digit recognitions. For the classification model, a custom model was implemented. It has three layers that have 10 neurons each. Relu Activision funciton is used. The softmax function is used in the last layer to get the probability distribution of the output predictions. It was also used in the top-k neuron prioritizing score to normalize output of each layer.

B. *Efficiency Metric*

In the ideal scenery, all error-revealing inputs are found in the first test cases. It was evaluated in all experiments so that it can be compared with the experimental results. Theoretically, the random case should be a straight line that has a formula of x=y. Since it is almost impossible to find the true random case, we evaluated the shuffled test results to simulate the random case. A line that covers more area than the random cases means, the evaluated method has sorted test cases better than the random cases, and error-revealing test cases could be found more quickly compared to the random case. The closer the line to the ideal case, the more successful the approach was.

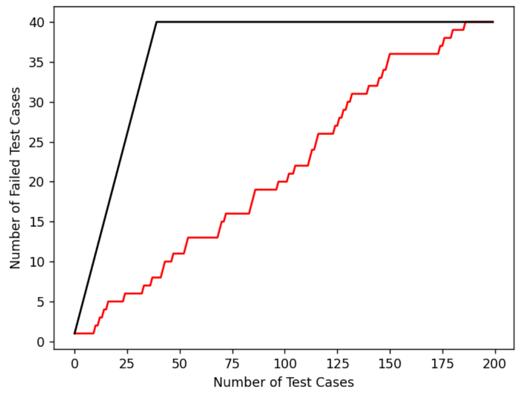

Fig. 3: Result for the ideal and random case. The black line represents the ideal case and the red one represents the random cases.

C. *Results*

We evaluated all the proposed methods on the custom model with the MNIST dataset. We also implement and add Byun's softmax-based method for comparison.

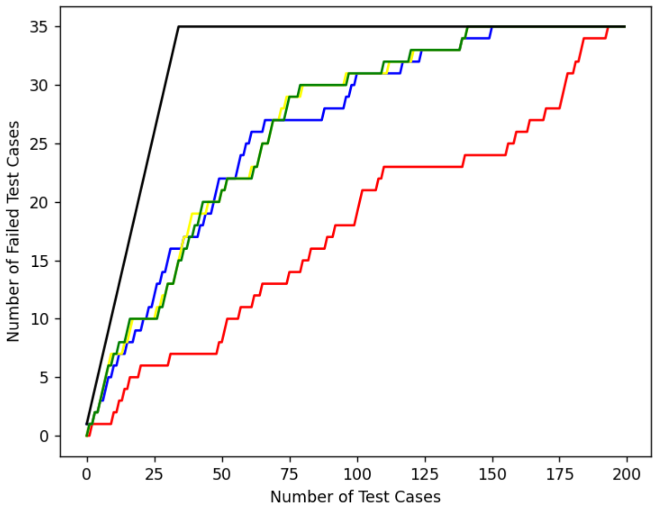

Fig. 4: The black line represents the ideal case, the red one represents the random cases, the blue one represents the Byun's method, the yellow one represents the standard deviation and the green one represents the l2 distance-based score. Model testing accuracy 82.5 %

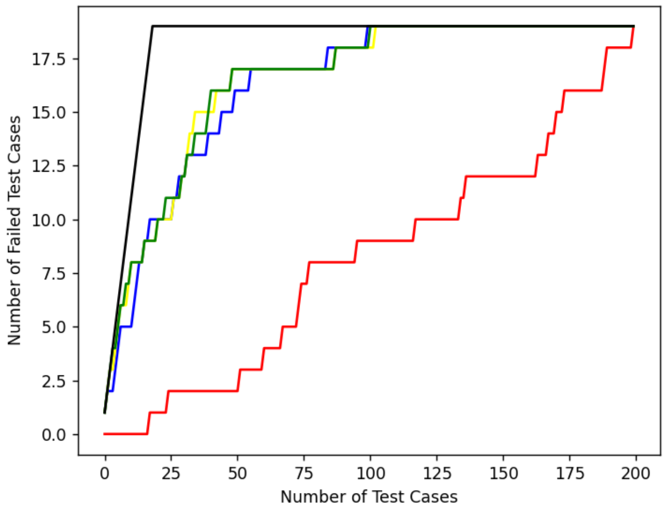

Fig. 5: The black line represents the ideal case, the red one represents the random cases, the blue one represents the Byun's method, the yellow one represents the standard deviation and the green one represents the l2 distance-based score. Model testing accuracy 90 %

It was observed that in the softmax-based method all the methods demonstrate a great result. Although all the softmax methods got very close results, standard derivation and l2 distance-based methods slightly perform better compared to Byun's softmax method. Also, it can be observed that these methods work better on the model with higher accuracy. In the 90% accuracy case, they found all error-revealing cases in almost half of the test cases.

On the other hand, although the proposed methods are promising, the result of these methods was not very successful. Both k=2 and k=3 cases are far from the ideal case. However, this method is still better than the random case, and it may be useful on different or more complex models.

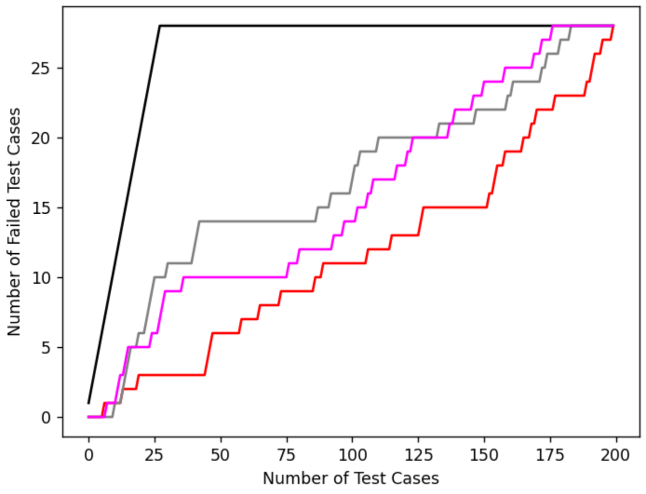

Fig. 6: The black line represents the ideal case, the red one represents the random cases, the purple one represents the k=2 most active neurons case and the grey one represents the k=3 most active neurons case. Model testing accuracy 86 %

D. Future Work

As future work, models with more complex architecture should be tried with these methods to evaluate their efficiency among them to see whether these methods are still valid or not for them. The top-k neuron prioritizing metric also should be tried on different architectures. Even though it did not show great success in our architecture, it still could show more efficient results from different architectures. Also, different neuron coverage methods could be used to create different prioritizing metrics.

V. Conclusion

In conclusion, we defined different input prioritizing metrics for reducing the labeling effort by finding all error-revaluing cases from the test dataset. These methods are evaluated in the experiments and results were discussed. The experiments demonstrated that most of the proposed methods successfully prioritize the test input among the test dataset.

References

[1] Taejoon Byun, Vaibhav Sharma, Abhishek Vijayakumar, Sanjai Rayadurgam, Darren Cofer, Input Prioritization for Testing Neural Networks, 2019

[2] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, Zhenyu Chen , DeepGini: Prioritizing Massive Tests to Enhance the Robustness of Deep Neural Networks, 2019

[3] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, Yadong Wang ,DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems, 2018

[4] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong and W. Zhang, Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis, 2021

[5] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, M. Shahzad, W. Yang, R. Bamler, X. X. Zhu, A Survey of Uncertainty in Deep Neural Networks, 2022