2019 IEEE/IFIP International Conference on Dependable Systems and Networks Workshops

NV-DNN: Towards Fault-Tolerant DNN Systems with N-Version Programming

Hui Xu*§, Zhuangbin Chen*, Weibin Wu*, Zhi Jin†, Sy-Yen Kuo‡, Michael R. Lyu*§
* Department of Computer Science and Engineering, The Chinese University of Hong Kong
† Key Laboratory of High Confidence Software Technologies (MoE), Peking University
‡ Department of Electrical Engineering, National Taiwan University
§ Shenzhen Research Institute, The Chinese University of Hong Kong

***Abstract*—Employing deep learning algorithms in real-world applications becomes a trend. However, a bottleneck that impedes their further adoption in safety-critical systems is the reliability issue. It is challenging to develop reliable neural network models as the theory of deep learning has not yet been well-established and neural network models are very sensitive to data perturbations. Inspired by the classic paradigm of N-version programming for fault tolerance, this paper investigates the feasibility of developing fault-tolerant deep learning systems through model redundancy. We hypothesize that if we train several simplex models independently, these models are unlikely to produce erroneous results for the same test cases. In this way, we can design a fault-tolerant system whose output is determined by all these models cooperatively. We propose several independence factors that can be introduced for generating multiple versions of neural network models, including training, network, and data. Experimental results on MNIST and CIFAR-10 both verify that our approach can improve the fault-tolerant ability of a deep learning system. Particularly, independent data for training plays the most significant role in generating multiple models sharing the least mutual faults.**

I. Introduction

Deep neural networks (DNNs) bring breakthroughs in many application fields, such as computer vision and natural language processing. It has become one of the most favorite techniques being explored by organizations to improve their systems or products. However, people also realize that DNN models are unreliable. On the one hand, they are vulnerable to data perturbations [3], [14]. On the other hand, even if some erroneous cases are known in advance, it is challenging to fix these issues by fine-tuning the model without introducing new faults.

Since theoretically guaranteeing the reliability of a single model remains difficult nowadays, we focus on promoting the fault-tolerant ability of deep learning systems with redundant models. Our idea is inspired by the classic N-version programming (NVP) paradigm [1], which creates several independent versions of software based on the same requirements. The core assumption of NVP is that we can minimize the chances that different versions of software share the same faults by maximizing the independence of their development processes. As DNNs are also a special kind of software[1], we can minimize the probability that different models all produce erroneous results for identical error-prone inputs.

[1]Software 2.0: https://medium.com/@karpathy/software-2-0-a64152b37c35

Nevertheless, we cannot directly apply classic NVP to deep learning models because the development of DNNs dramatically differs from that of traditional software. Note that the logic of deep learning models is automatically learned from data instead of explicitly being coded by developers. Therefore, it is unnecessary to manually program independent model versions which induces large programming cost. While this characteristic brings an advantage of our approach over traditional NVP, it also introduces a new challenge, *i.e.,* how to achieve independence while creating these models so that the final system can achieve better fault-tolerant performance?

To address the challenge, we identify the major independence factors that can be introduced for generating different versions of models, including *independent training*, *independent network*, and *independent data*. While designing different networks and preparing different training data require human intervention, independent training can be fully automated. We empirically study the effectiveness of these factors with two popular image recognition datasets, *i.e.,* MNIST and CIFAR-10. Experimental results show that all these factors are effective to improve the fault-tolerant ability of a deep learning system. However, independent data outperforms independent network, and independent training is the least effective.

II. Related Work

A well-recognized approach related to our work is neural network ensemble [4], which is based on the fact that there could be multiple local optima obtained when training a neural network, leading to different outputs for the same error-prone input. The approach in nature employs several models with different local optima and a collective decision algorithm, aiming at achieving better accuracy. Traditionally, the idea interests the scholars of machine learning fields, and there has been some recent progress achieved, such as MC-DNN [2] and EC-DNN [10].

In comparison, our work is based on a different assumption that independently developed neural network models would behave differently to the same error-prone input. It emphasizes the independence of the software development process, while does not restrict the optimization problem to be the same one. In practice, independence could be introduced in any phases of the software development process, such as data collection,

978-1-7281-3030-9/19/$31.00 ©2019 IEEE
DOI 10.1109/DSN-W.2019.00016

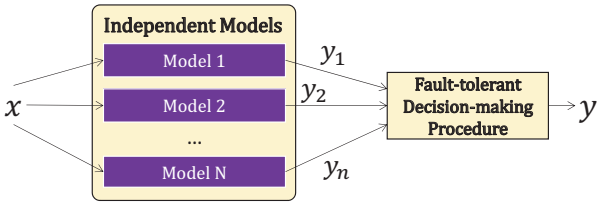


Fig. 1. The conceptual framework of NV-DNN.

network design, and training. For example, employing different network structures or data to train multiple models involve different optimization problems in nature rather than merely different local optima. Furthermore, the fault-tolerant mechanism based on multiple models could be very complicated and application-specific.

Besides, there are other investigations towards improving the robustness of neural networks, such as feature squeezing [13] and deep validation [12]. Since they differ a lot from our work, we do not go into details. Our approach is orthogonal to most of these approaches, and it can be integrated with them to provide better reliability.

III. N-version Deep Learning

This section discusses the concept and approach of applying N-version programming to designing fault-tolerant DNN systems, namely NV-DNN.

A. Concept and Metrics

Figure 1 demonstrates the conceptual framework of NV-DNN. It contains N independently developed models and a decision procedure. By introducing independence or variance in the model development process, NV-DNN aims to generate multiple simplex models which share as less mutual faults as possible. The decision procedure thus can compute the final result based on these models and achieves fault tolerance.

The key challenge for NV-DNN is to ensure that the residual faults of each model are unique, *i.e.,* identical faults should not occur in multiple versions. Otherwise, the faults may not be tolerable for the entire system. We can employ *distinguished error rates* (DER) to evaluate the effectiveness of a solution regarding the challenge. DER measures the rate of erroneous test cases occurred in only one model over all the erroneous cases of individual models.

$$DER(\{M_1, ..., M_n\}) = \frac{\#\text{ of unique erroneous cases}}{\#\text{ of all erroneous cases}}. \quad (1)$$

A higher DER implies more faults can be tolerated with NV-DNN. It reflects the lower bound of the number of faults that are tolerable. A fault can be tolerable if it is unique, such as with a voting-based decision-making algorithm. However, this is not a necessary condition. Another metric that can reflect the upper-bound fault-tolerant capability is *mutual error rate* (MER), which is the rate of erroneous cases shared by all models.

$$MER(\{M_1, ..., M_n\}) = \frac{\#\text{ of mutual erroneous cases}}{\#\text{ of all erroneous cases}}. \quad (2)$$

In general, mutual faults are not tolerable. An NV-DNN solution is better with a higher DER and a lower MER.

B. Generating Independent Models

To generate independent models, there are several independence factors that can be introduced in the DNN development process, such as independent training, independent network, and independent data.

1) Independent Training: Introducing randomness in the training process of DNNs is a standard practice [8]. For example, the parameters of models are randomly initialized following a pre-specified distribution, and training data are shuffled before being fed to the network. Therefore, independent training process can lead to different local optima or different models. Besides, there are other random factors which can also affect the optimization process, such as learning rate and optimizer. Since these factors do not change the optimization problem, we categorize all of them as default randomness or independent training.

2) Independent Network: We can design different network architectures and let them learn different features. Such structural differences or independence can be minor ones (*e.g.,* replacing max pooling operators with average pooling) or major ones (*e.g.,* adopting different neural network structures like GoogLeNet [11] and ResNet [5]). Since NVP prefers more independence of the development processes, we think employing major structural differences may perform better in fault tolerance. However, it should guarantee that each network can achieve competent accuracy.

3) Independent Data: We may employ different corpora of data about the same problem to train each simplex model. Such differences can change the optimization problems and provide more diversity. For example, we may split a large dataset into several subsets. If the resources of data are limited, we may also adopt variant data augmentation techniques to generate a unique set of training data for each simplex model.

C. Decision Procedure

With multiple models for NV-DNN, the decision procedure computes a final result based on their outputs. A simple decision procedure could be voting-based, which is favored by existing model ensemble approaches. For example, we can sum up the probabilities of each label outputted by these models and choose the label with the maximum probability as the final result.

Furthermore, since NV-DNN is a fault-tolerant solution, it may employ more sophisticated decision procedures to tolerate faults, such as failover. Because the mechanism could be application-specific, we leave such discussions as our future work.

TABLE I
EVALUATION RESULTS. INDEPEN: INDEPENDENCE, M-ACC: MODEL ACCURACY, IMP-ACC: IMPROVED ACCURACY OF NV-DNN.

Experimental Setting				MER	DER	Imp-Acc
Dataset	Indepen.	Network	M-Acc			
MNIST	Training	LeNet5	99.57%	0.17	0.55	99.63%
			99.56%			
			99.55%			
	Network	LeNet5	99.57%	0.21	0.59	99.63%
		LeNet5A	99.57%			
		LeNet5B	99.58%			
	Data	LeNet5	99.57%	0.17	0.62	99.67%
			99.60%			
			99.56%			
CIFAR -10	Training	ResNet	91.67%	0.24	0.49	92.99%
			90.94%			
			91.25%			
	Network	ResNet	91.67%	0.20	0.52	93.10%
		GoogLeNet	91.50%			
		DenseNet	90.14%			
	Data	ResNet	91.67%	0.20	0.54	93.21%
			90.26%			
			90.67%			

IV. EXPERIMENTS

This section presents our experimental study. Firstly, we compare the effectiveness of independence factors for generating multiple models. Secondly, we examine the feasibility of NV-DNN for improving fault-tolerant ability.

A. Experimental Setting

We choose two popular datasets for evaluation: MNIST [2] and CIFAR-10 [3]. For MNIST, we employ LeNet-5 [9] as the default network. To examine the performance of independent networks, we change the pooling layers of LeNet-5 and obtain two variant networks, LeNet5-A and LeNet5-B. For CIFAR-10, we choose GoogLeNet [11], ResNet [5], and DenseNet [6] as the networks.

When evaluating the effectiveness of independent data for NV-DNN, we use data augmentation to generate two more datasets. In particular, we apply random rotations to the original images to produce a new dataset. For MNIST, we randomly adjust the brightness and contrast to generate another dataset. For CIFAR-10, we further modify the saturation to generate the third dataset.

The experiments are conducted with PyTorch 1.0. We train each simplex model with Adam optimizer [7] and the recommended learning rate. For MNIST, we train 100 epochs for each model; for CIFAR-10, we train 1000 epochs.

B. Experimental Results

Table I presents our experimental results. Overall, the results show that NV-DNN is effective in promoting system fault tolerance, and adjusting networks and training data perform better than merely relying on the default randomness.

[2]http://yann.lecun.com/exdb/mnist/

[3]https://www.cs.toronto.edu/kriz/cifar.html

TABLE II
EVALUATION RESULTS WITH SEVEN INDEPENDENT MODELS.

Experiment	Mutual Error #	Accuracy
MNIST	6	99.69%
CIFAR-10	147	93.76%

1) Factor Comparison: We compare the effectiveness of different factors using DER and MER. A factor is superior to the other if it can achieve a higher DER and a lower MER. We do not compare the fault-tolerant ability directly because they are related to the choice of decision-making procedures.

For the results of MNIST, the factor of independent data clearly outperforms the other two. However, the performance differences between independent training and independent network are not obvious. We suspect a major reason is that MNIST is too simple, and our models have already achieved very high accuracy. As a result, we do not have many faults to compare, and the result may not have statistical significance.

The results of CIFAR-10 are more obvious, which show that independent data outperforms independent network, and independent training is the least effective factor.

2) Effectiveness of Fault Tolerance: To evaluate the fault-tolerant ability, we study the achieved accuracy improvement of the NV-DNN system with a simple voting-based decision procedure. We average the output of each simplex model as the final result and evaluate the NV-DNN model on the original testing dataset.

According to our experimental results, the accuracy on MNIST is 99.63% for independent training, 99.63% for independent network, and 99.67% for independent data. All these results are better than the original models. For CIFAR-10, the corresponding accuracy is 92.99% for independent training, 93.10% for independent network, and 92.31% for independent data, respectively. The accuracy improvements are very obvious.

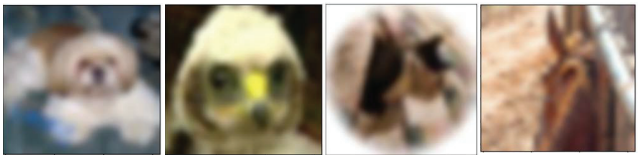
C. Discussion

Our results show that NV-DNN is effective in promoting fault tolerance. The accuracy can get improved when multiple independent models are applied. Previously, we only examined the effectiveness of three simplex models. Can we further improve accuracy when employing more versions? To answer this question, we conduct another experiment with seven independent models and re-evaluate the accuracy. Our experiment is based on the same models discussed in Table I. There are nine models for each dataset, but the first model of each independence experiment is identical. So our experiments employ seven unique models for each task. Table II presents the results, which show that the accuracy can get further improved to 99.69% for MNIST and 93.76% for CIFAR-10. These results verify the possibility of further improving the fault-tolerant ability with more models.

However, we also find there are some erroneous cases mutual to all models. The numbers of such cases are 6 and 147 for MNIST and CIFAR-10, respectively. These faults

(a) Mutual erroneous cases of MNIST. The labels in the original dataset are 9, 9, 5, and 6 respectively.

(b) Erroneously labeled cases of the MNIST dataset. The corresponding labels in the original dataset are 0 and 6, which should be 6 and 4 from our perception.



(c) Mutual erroneous cases of CIFAR-10. The labels in the original dataset are dog, bird, cat, and horse respectively.

Fig. 2. Demonstration of the failed test cases.

can hardly be addressed even with more models or advanced decision-making procedures. Nevertheless, is it possible to manually classify these images? Figure 2 demonstrates several such samples extracted from the original datasets. For the erroneous cases of MNIST in Figure 2(a), our models falsely recognize them as 4, 4, 3, and 0 respectively. In reality, it is even challenging for us to recognize them correctly, and we tend to make similar mistakes as the models do. Besides, we find several images should be erroneously labeled in the original dataset. Figure 2(b) demonstrates two examples which our models tend to make mistakes. Their original labels are 0 and 6 respectively in the dataset. But we think they should be 6 and 4 from human perception. The images of the CIFAR-10 dataset (Figure 2(c)) do not have good resolutions, so it is difficult for human perception. For instance, the face of the first image is very similar to that of the second one. We can hardly recognize whether it is a dog or a bird. The third image is even vaguer.

On the other hand, these failures also imply there is still room to improve our current NV-DNN systems. We believe collecting more training data to train independent models should be a promising way. The main reason is that increasing the diversity of training data can enlarge the explored regions of the large feature space. Moreover, our experimental results have shown that the factor of independent data is more effective than others. If we can collect similar samples that an NV-DNN system failed and train a new model based on such samples, the model may be able to produce the correct result. In our NV-DNN solution, we can also train independent models for specific error-prone cases, such as the digits shown in Figure 2(a). However, it requires a more sophisticated decision-making procedure to integrate different models. We leave such problems as our future work.

V. Conclusion

This paper is a preliminary investigation towards developing reliable deep learning software with N-version programming. An advantage of NV-DNN over traditional N-version programming is that it costs less to generate multiple versions because deep learning models are automatically trained. We have introduced the concept of NV-DNN and discussed several independent factors for generating different model versions, including independent training, independent network, and independent data. Experimental results show that all these factors are effective for fault tolerance, and employing independent training data can achieve the best performance. There are several interesting directions which could be investigated in the future, especially some advanced approaches for developing independent models and decision-making procedures. Besides, investigating the resilience of NV-DNN to unintentional data perturbations and various attacks is also an important task.

References

[1] A. Avizienis. The methodology of n-version programming. *Software Fault Tolerance*, 1995.

[2] D. Cireşan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[3] A. Fawzi, S. M. Moosavi Dezfooli, and P. Frossard. The robustness of deep networks-a geometric perspective. *IEEE Signal Processing Magazine*, 2017.

[4] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 1990.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 1998.

[10] S. Sun, W. Chen, J. Bian, X. Liu, and T.-Y. Liu. Ensemble-compression: A new method for parallel training of deep neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[12] W. Wu, H. Xu, S. Zhong, M. Lyu, and I. King. Deep validation: Toward detecting real-world corner cases for deep neural networks. In *Proc. of the 49th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[13] W. Xu, D. Evans, and Y. Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *The Network and Distributed System Security Symposium (NDSS)*, 2018.

[14] X. Yuan, P. He, Q. Zhu, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2019.