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

Anonymous Submission for Double-blind Review

***Abstract*—Employing deep learning algorithms in real-world systems 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 models as the theory of deep learning algorithms has not yet been well-established, and deep learning 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 designing 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 identify the major factors that can be adjusted for generating different versions of deep learning models, including the default randomness, network structures, and training data. Then we conduct real-world experiments to evaluate the feasibility of our approach and compare the effectiveness of these factors for improving the fault tolerance of deep learning systems. Our results on MNIST and CIFAR-10 both verify that this approach can effectively reduce the failure rate of models for inference. Particularly, the factor of training data plays the most significant role in generating different model versions in order to reduce the failure rate.**

## 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], [15]. On the other hand, even if the erroneous cases are known in advance, it is challenging to fix the issue 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 share the same faults by maximizing the independence of their development processes. As DNNs are also a special kind of software, we can minimize the probability that different models all produce erroneous results for the identical cases.

Nevertheless, we cannot apply classic NVP to deep learning models directly because the development of DNNs dramatically differs from that of traditional software. Note that the logic of deep learning models are automatically learned from data instead of explicitly being coded by developers. Therefore, it is unnecessary to manually program different versions of models with high programming cost. While this characteristic brings an advantage over traditional NVP, it also introduces a new challenge, *i.e.,* what factors should we adjust to create these models so that the final system can achieve better fault-tolerant performance?

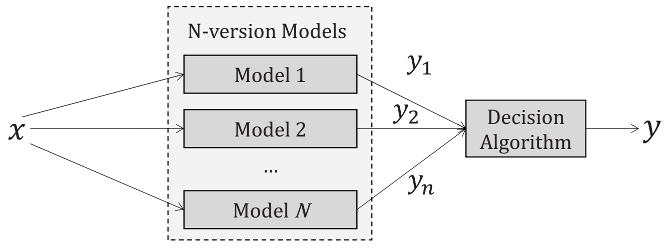


Fig. 1. The conceptual framework of N-version deep learning.

To address the challenge, we first identify the major factors for generating different versions of models, including *default randomness*, *network structures*, and *training data*. While designing different networks and preparing different training data require human intervention, default randomness (*e.g.,* model initialization and optimization process) is automatically introduced into the training process by default. Then we empirically study the effectiveness of these factors with two popular image recognition tasks, *i.e.,* MNIST and CIFAR-10. Experimental results show that all these factors are effective to reduce the resulting failure rate of the deep learning system. However, employing different training data and network structures can achieve better results than the weak default randomness.

## II. 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 $N$ simplex models and a decision making algorithm. By introducing variances in the model designing and training phase, NV-DNN aims to generate multiple simplex models which share as less mutual faults as possible. In this way, the decision making algorithm can compute the final result based on the outputs of these models.

The key challenge for NV-DNN is to ensure that the residual faults of each model is unique, *i.e.,* the 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 with voting-based decision making algorithms. A NV-DNN solution is better with a higher DER and a lower MER.

## B. Generating Simplex Models

To generate diversified simplex models, there are several factors that can be introduced into the DNN development process, such as default randomness, network difference, and data difference.

*1) Default Randomness:* Introducing some randomness in the training process of DNNs is a standard practice [7]. 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. These factors can lead to different local optimums, *i.e.,* different models. Besides, there are other 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.

*2) Network Difference:* We can design different network architectures and let them learn different features. Such structural differences 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 [13] and ResNet [4]). 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 a competent accuracy.

*3) Data Difference:* 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 Making Algorithms

With multiple models for NV-DNN, the decision making algorithm can compute a final result based on their outputs. For regression problems, the result could be an average of their outputs. For classification problems, the result could be voting-based. For example, we can sum up the probabilities of each label outputted by these the models and choose the label with the maximum probability as the final result.

## III. Experiments

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

## A. Experimental Setting

We choose two popular datasets for evaluation: MNIST [1] and CIFAR-10 [2].

We choose LeNet-5 [8] as the default network for MNIST. To examine the performance of different networks, we change the pooling layers of LeNet-5 and obtain two variant networks, LeNet5-A and LeNet5-B. We do not adopt LeNet-1 and LeNet-4 because their accuracies are obviously lower than that of LeNet-5 [9]. For CIFAR-10, we choose GoogLeNet [13], ResNet [4], and DenseNet [5] as the networks.

When evaluating the effectiveness of data difference for NV-DNN, we use data augmentation to generate multiple datasets. In particular, we apply random rotations to the original images to produce a new dataset. For MNIST, image transformations include randomly adjustments on the brightness and contrast, while for CIFAR-10, besides the above two properties, saturation is also modified to encourage dataset randomness.

The experiments are conducted with PyTorch 1.0. We train each simplex model with Adam optimizer [6] and the recommended learning rate. For MNIST, we train 100 epochs for each model; for CIFAR-10, we train 1000 epochs. Because we have to prepare multiple version of models, we do not directly employ the models available on the internet.

## 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.

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

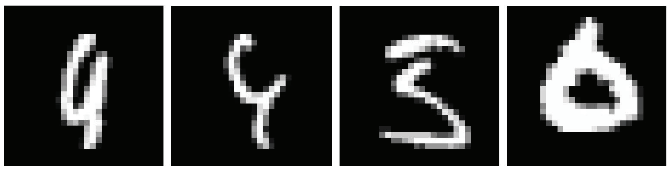
[2]https://www.cs.toronto.edu/kriz/cifar.html

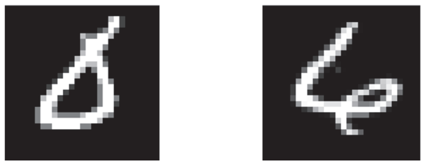
TABLE I
EVALUATION RESULTS. MOD-ACC: MODEL ACCURACY, IMP-ACC: IMPROVED ACCURACY OF NV-DNN.

| **Experimental Setting** | | | | **MER** | **DER** | **Imp-Acc** |
|---|---|---|---|---|---|---|
| Dataset | Factor | Network | Mod-Acc | | | |
| MNIST | Random | 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 | Random | 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% | | | |

TABLE II
EVALUATION RESULTS WITH SEVEN MODELS.

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


(a) Mutual erroneous cases of MNIST. The corresponding labels are 9, 9, 5, and 6.


(b) Erroneously labeled cases of the MNIST dataset. The corresponding labels in the original dataset are 0 and 6.

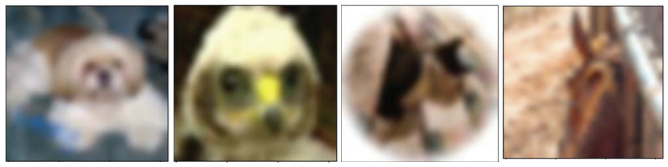

(c) Mutual erroneous cases of CIFAR-10. The corresponding labels are dog, bird, cat, and horse.

Fig. 2. Demonstration of the failed test cases.

*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 failure rates directly because they are related to the choice of decision making algorithms.

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

The results of CIFAR-10 are more obvious, which show that data difference outperforms network difference, and default randomness is the least effective.

*2) Effectiveness of Fault Tolerance:* To evaluate the fault-tolerant ability, we study the achieved accuracy improvement of the NV-DNN model. We average the output of each simplex model as our decision making algorithm and evaluate the NV-DNN model on the original testing dataset.

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

## C. Discussion

Our results show that NV-DNN is effective in promoting system fault tolerance. The accuracies can get improved when multiple models are applied. Previously, we only examined the effectiveness with three simplex models. Can we further improve the accuracy when employing more versions? To answer this question, we conduct another experiment with seven unique models and evaluate the accuracies again. 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 factor is identical. So our experiments employ seven unique models for each dataset.

Table II shows our results. When combining the seven models together, the accuracies get further improved to 99.69% for MNIST and 93.76% for CIFAR-10. The results confirm 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 can hardly be addressed even with more models or advanced decision making algorithms. 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 there are several images that 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 similar to that of the second one. We can hardly recognize whether it is a dog or a bird. The third image is more vaguer.

On the other hand, these failures also imply there is still a room to improve our current NV-DNN models. We believe collecting more training data to train different 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 training data is more effective than others. If we can collect similar samples that a NV-DNN 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 models for specific error-prone cases, such as the digits shown in Figure 2(a). However, it requires a more sophisticated decision making algorithm to integrate different models. We leave such problems as our future work.

## IV. Related Work

There are already some efforts towards improving the reliability of neural networks. While one group focuses on training models with better accuracy (*e.g.,* [2], [12]), the other focuses on the resilience to adversarial samples (*e.g.,* [10], [14]). Next, we discuss these ideas that also employ multiple models.

Cirecsan *et al.* [2] propose multi-column deep neural networks (MC-DNN), which is an ensemble-based approach for deep learning. They employ the average outputs of local models as the final result. Similarly, Sun *et al.* [12] propose EC-DNN, which improves the training process and compresses the models to further improve the efficiency. To our best knowledge, such papers employ the same network architecture and dataset to train multiple local models. NV-DNN shares similarity with model ensemble, but it focuses on a fault-tolerant perspective. Our research aims to minimize the mutual error rate. Moreover, it may pursue a complicated decision procedure.

Xu *et al.* [14] propose feature squeezing, which improves the model resilience to adversarial examples leveraging multiple data preprocessing approaches or squeezers during inference. It firstly predicts the results based on each squeezer and then employs a decision making algorithm to determine the final results. We also employ different data preprocessing techniques which is in the training phase. Feature squeezing is an approach orthogonal to NV-DNN. Besides, Srisakaokul *et al.* [11] proposed multi-implementation testing, which leverages multiple machine learning models to generate test oracles. It is a fault-prevention approach in software development phase instead of fault tolerance in execution phase.

## V. Conclusion

This paper investigates the reliability of deep learning models from the fault-tolerant perspective. We propose to apply the idea of the classic N-version programming to build deep learning models with improved fault-tolerant ability. An advantage of NV-DNN over traditional N-version programming is that it costs much less to generate multiple versions because deep learning models are automatically trained. We have introduced the concept of NV-DNN and discussed the approaches to build a NV-DNN model. We also discussed factors that can be adjusted in order to generate different versions of models, which include default randomness, network difference, and data difference. We have conducted several experiments, and results show that all these factors are effective for fault tolerance, and the NV-DNN models with different training data achieved the best performance.

## 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. *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] 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.

[5] 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.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[9] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*. Perth, Australia, 1995.

[10] K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated whitebox testing of deep learning systems. *Proc. of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[11] S. Srisakaokul, Z. Wu, A. Astorga, O. Alebiosu, and T. Xie. Multiple-implementation testing of supervised learning software. In *Proc. AAAI-18 Workshop on Engineering Dependable and Secure Machine Learning Systems (EDSMLS)*, 2018.

[12] 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.

[13] 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.

[14] W. Xu, D. Evans, and Y. Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. 2018.

[15] 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*, 2019.