Fault Tolerance in Triplet Network Training: Analysis, Evaluation and Protection Methods

Ziheng Wang, *Student Member, IEEE*, Farzad Niknia, *Student Member, IEEE*, Shanshan Liu, *Senior Member, IEEE*, Pedro Reviriego, *Senior Member, IEEE*, Ahmed Louri, *Fellow, IEEE* and Fabrizio Lombardi, *Life Fellow, IEEE*

***Abstract*—This paper investigates the tolerance of Triplet Networks (TNs) with a focus on faults in the training process. For compatibility with the existing literature. So-called stuck-at faults of a functional nature are considered for the operation of the neurons and activation function. While TNs are shown to be generally robust against such faults in the anchor and positive subnetworks, the presented analysis reveals a significant vulnerability in the negative subnetwork, in which stuck-at faults can lead to false convergence and training failures. An in-depth treatment is provided to show the incorrect convergence of training in the presence of stuck-at faults, highlighting the behavior of the network with faulty neurons. Extensive simulations are presented to evaluate the impact of these faults and propose two innovative fault-tolerant methods: the regularization of the anchor outputs and the modified margin. Simulation shows that false convergence can be very efficiently avoided by utilizing the proposed techniques, and thus the overall accuracy loss of the TN is negligible. These findings contribute to the understanding of fault tolerance in emerging neural networks such as TNs and offer practical solutions for enhancing their robustness against faults.**

***Index Terms*— Triplet network, fault-tolerance, stuck-at faults, training, false convergence.**

I. Introduction

TRIPLET networks (TNs) are feedforward artificial neural networks (ANNs) with three identical weight-sharing subnetworks [1]. TNs have been widely used in similarity-measuring tasks, from initial applications in face recognition with Triplet Loss (TL) [2] to various machine learning (ML) tasks such as vehicle identification [3] and image retrieval [4]. Particularly, due to the unique structure, TNs provide an excellent learning performance when there is a paucity of training data available, in which case the traditional ANNs using a single network (e.g., multi-layer perceptrons (MLPs) or convolutional neural networks (CNNs)) often find difficulty in establishment and execution. A TN generates feature embeddings of the original input data; the outputs of different categories are separable in the embedding space and thus, are available for the subsequent classification/recognition. As an emerging ML scheme, the feature-embedding TN achieves better performance than traditional classification/recognition methods with a single network branch [1], [5].

In critical safety applications, the robustness of ANNs is paramount, because faults can degrade performance or even cause a complete system failure [6]-[9]. When analyzing the fault tolerance of ANN training, the stuck-at fault model is usually used; this model is not physically based but it functionally abstracts the behavior of defective neurons [7], [10]. Although past literature extensively covers this model and its effects, current investigations often focus on smaller, simpler structures, e.g., three-layer MLPs with a maximum of 20 neurons in the hidden layer [11]-[15]. However, today's ANNs are significantly larger and incorporate substantial overprovisioning, allowing for sustained performance despite some non-functional neurons [7], [16]. The fault tolerance of large-scale networks such as TNs has not been sufficiently explored in the technical literature.

Moreover, the distinctive structure of three weight-sharing subnetworks presents unique challenges for attaining fault tolerance of TNs. Its convergence condition aims to minimize the output distances between the anchor and the positive subnetwork, and maximize it between the anchor and the negative subnetwork. In the fault injection experiments (such as faults in the positive or anchor subnetworks) conducted in this paper, extensive faults could violate the convergence condition, resulting in a performance degradation; however, stuck-at faults in the negative subnetwork lead to rapid but erroneous convergence, evading detection and potentially causing a fatal system failure. Therefore, it is essential to analyze and protect the TNs from such a special case, that to our knowledge, has not been addressed in the existing literature.

This paper initially addresses the general sensitivity of TNs to stuck-at faults; furthermore. It then conducts an in-depth investigation of the specific cases that lead to false convergence caused by stuck-at faults in the negative subnetwork. Unlike situations with a small fault impact, the deceptive convergence leads to substantial degradation and causes system failures. Alongside the analysis and evaluation of the faults affecting the classification performance of TNs, this paper also proposes efficient protection methods against false convergence. In practical applications, the scheme allows TNs for failure-free training (and re-training) even with faulty hardware, which helps increase the lifetime of the circuit in such cases.

The contributions of this paper are as follows:

1) The impact of stuck-at faults in the positive and anchor subnetworks of a TN is comprehensively evaluated by considering different positions, rates, and types of faults.

Manuscript received May 16, 2024 and revised August 29, 2024. The work was supported by the NSF under Grant 1953961 and Grant 1812467 and by the Spanish Agencia Estatal de Investigación under Grants PID2022-136684OB-C22 and PDC2022-133888-I00 (*Corresponding author: Shanshan Liu*).

Ziheng Wang, Farzad Niknia and Fabrizio Lombardi are with Department of Electrical and Computer Engineering, Northeastern University, MA 02115, USA.

Shanshan Liu is with School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, China (email: ssliu@uestc.edu.cn).

Pedro Reviriego is with the Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, 28040 Madrid, Spain.

Ahmed Louri is with the Department of Electrical and Computer Engineering, George Washington University, DC 20052, USA.

2) A focused analysis is presented for the false convergence caused by stuck-at faults in the negative subnetwork. Three convergence results are categorized by their impact on the training process.
3) A further evaluation on the impact of faults in the negative subnetwork with different influence factors is conducted. Results confirm the analytic impact, showing that the accuracy can be significantly degraded due to training failures with a false convergence.
4) Two schemes are proposed to deal with faults in the negative subnetwork, avoiding deceptive convergence. Evaluation results show that the accuracy loss of TNs is negligible when applying these protection schemes.

The rest of the paper is organized as follows. Section II outlines preliminaries, such as the structure of TNs and the fault model considered in this paper. Section III evaluates the impact of faults in the positive and anchor subnetworks of the TNs, while Section IV deals with the faults in the negative subnetwork. Both analysis and evaluation of the false convergence caused by the faults are presented in Section IV. To mitigate the impact of these faults and avoid false convergence, two fault-tolerant schemes are proposed in Section V. The limitations to the applicability of the analysis and proposed protection schemes for specific implementations are discussed in Section VI; finally, the paper ends with the conclusion in Section VII.

II. Preliminaries

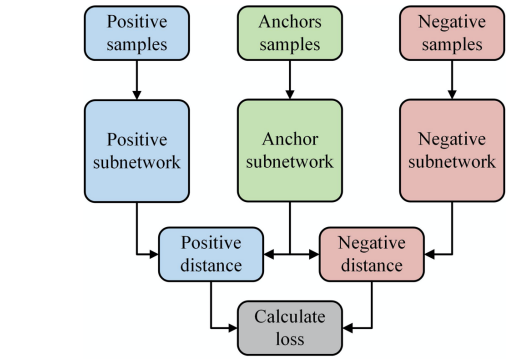
A. Triplet Networks and the Training Process

A typical Triplet Network (TN) consists of three weight-sharing subnetworks in training as shown in Fig. 1 (a). Each subnetwork can be any feedforward network; however, considering the nature of the three weight-sharing subnetworks in a TN, the subnetwork of most TNs is usually implemented using MLPs and CNNs due to their lower model complexity and hardware overhead, while larger models may not be applicable[1]. This paper focuses on the application of deep metric learning classification of TNs and it showed excellent performance with a paucity of training data [5].

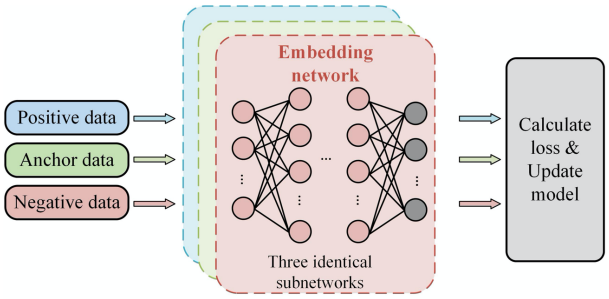
During the training process (Fig. 1 (b)), the three subnetworks of a TN accept the so-called triplets, including anchors, positive samples within the same class, and negative samples within different classes, respectively. The model is trained by mapping the triplets to an embedding space to minimize the distances between the anchor and positive samples and maximize the distance between the anchor and negative samples. Such feature embeddings are learned by iteratively applying optimizers (such as gradient descent) to the loss for a high classification performance [1], [5].

Assume that the subnetworks are l-layer MLPs (or CNNs), and the output of the hidden units in the r-th layer can be denoted as the following vectors

[1]This paper focuses on the MLP implementation in the main body and put the similar results for the CNN version in the Appendix. However, the analysis, findings and proposed protection techniques presented in this paper can also be applied to larger/newer models when they are used as the subnetworks of a TN.



(a)



(b)

Fig.1. A triplet network: (a) overall structure; (b) training process.

$$\begin{aligned} y_a^r &= \left[y_{a,1}^r \;\; y_{a,2}^r \;\; \cdots \;\; y_{a,m_r}^r\right], \text{ anchor subnetwork;} \\ y_p^r &= \left[y_{p,1}^r \;\; y_{p,2}^r \;\; \cdots \;\; y_{p,m_r}^r\right], \text{ positive subnetwork;} \\ y_n^r &= \left[y_{n,1}^r \;\; y_{n,2}^r \;\; \cdots \;\; y_{n,m_r}^r\right], \text{ negative subnetwork} \end{aligned} \quad (1)$$

where m_r is the output dimension of the r-th layer. In particular, the outputs of the last layer are represented as y_a^l, y_p^l, and y_n^l. The similarity is evaluated by the Euclidean distance, which is calculated by the positive distance $d_+ = \left\|y_a^l - y_p^l\right\|_2$, and the negative distance $d_- = \|y_a^l - y_n^l\|_2$. Since the objective of training a TN is to minimize the positive distance and maximize the negative distance, the following conditions must be satisfied:

$$d_+ = \sqrt{\sum_{i=1}^{m_r}\left(y_{a,i}^r - y_{p,i}^r\right)^2} < \varepsilon_+; \quad (2)$$

$$d_- = \sqrt{\sum_{i=1}^{m_r}\left(y_{a,i}^r - y_{n,i}^r\right)^2} > \varepsilon_- \quad (3)$$

where ε_+ and ε_- are constant margins. Moreover, the loss function L usually applies a relaxed constraint that optimizes the difference between the two distances $d_+ - d_-$ [20]. For example, this paper employs the popular Triple loss (TL) [2] and the trained model thus satisfies:

$$L_{TL} = max(d_+ - d_- + M, 0) \leq \varepsilon \quad (4)$$

where M is the margin and ε is a non-negative constant as the stopping criterion.

For the application of deep metric learning, an extra prediction network is required to map the embedded data to labels. Since the data from the same category has already been clustered in the embedding space by the TN, the prediction can be realized by simpler schemes such as Support Vector Machines (SVM), logistic regression, or K-Nearest Neighbors (KNN) [1]. This paper uses a one-hidden-layer MLP as the prediction network.

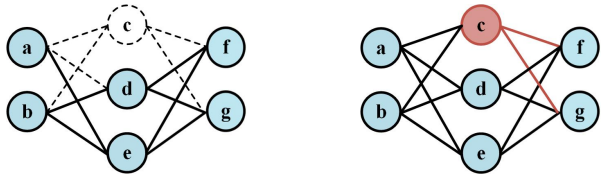
B. Fault Model and Datasets

As introduced previously, the unique features and benefits of a TN come from its special structure of the subnetworks that are employed during training. Moreover, the weight-sharing configuration is likely to make the subnetworks more sensitive to faults, degrading the performance of the trained model. Therefore, this paper focuses on investigating the fault tolerance of the training process in TNs.

The traditional stuck-at model describes faults and defects (so mostly at a physical level) by associating permanent logic values in devices (such as transistors) and circuits in Very Large Scale Integration (VLSI) designs and commonly occurring at manufacturing and operational time [6]. However, when analyzing fault tolerance in ANNs, fault models could be employed according to specific implementation details such as technology and behavioral specifications. These models are often mapped to high-level abstractions that are independent of their physical implementations [16]. For ANNs, the so-called stuck-at model has been widely reported in the literature [17]-[19], allowing incorrect behaviors that affect specific implementations to be represented by errors in the neurons [10]. Errors such as unexpected values, which may be caused by faulty/defective phenomena in ANNs, are typically abstracted using such stuck-at model. This model simplifies faults to show as constant values affecting individual hardware components (in this case the output of a neuron). This abstraction is extensively utilized in related works and has proven to be effective in modeling a substantial number of physical faults [16]. Based on the impact found in the technical literature [14], this functionally abstracted stuck-at model in ANNs is categorized into two types (Fig. 2): *stuck-at-zero* (where neurons are missing) and *stuck-at-max/min* (where neurons are saturated).

- ***Stuck-at-zero:*** A stuck-at-zero fault occurs when a neuron's output remains at zero regardless of the input. This means the neuron does not respond to any inputs, as if it were removed from the network. It can be caused by a variety of sources including physical defects, activation function saturation (such as ReLU), and weight loss.
- ***Stuck-at-max/min:*** A stuck-at-max/min fault occurs when a neuron's output no longer responds to changes in the input and is constantly given by the maximum or minimum value of its activation function. In addition to hardware faults, it usually happens when the activation function reaches its hard saturation limits (e.g., "tanh" and "sigmoid" have extreme values of 1 or -1).

As described above, there are multiple causes for the stuck-at model; this abstracted fault model enables the investigation of fault tolerance of ANNs at a behavioral level, independent of the specific intricacies of the implementation reflecting physical faults. So the (functional) stuck-at fault model utilized in this paper (as consistent with the technical literature) is not physically based; it allows to consider the functionality of the entire ANN (neuron and activation function). In this paper, the execution of the training process in TNs with specific faulty neurons is represented by patterns [13]. As the exhaustive testing for all possible fault-injection cases is prohibitive, for simplification purposes, this paper assumes that all faults are injected into the same layer and the same subnetwork. Even though a mathematical analysis is difficult for more comprehensive cases, our experiments demonstrate that faults have the same behavior when they extend across various layers and subnetworks and the false convergence still occurs as long as the negative subnetwork is affected. For the sake of simplicity, hereafter the pattern of faults used in the evaluation is defined as the tuple $Q_T = (f, r, c, v)$, where f is the number of faults and r denotes the layer to be injected (the input layer is not included), $f < m_r$; c is the faulty subnetwork, $c \in \{anchor, positive, negative\}$; v denotes the type of the corresponding faults, which means the faulty units stuck at v. This paper applies tanh as the activation function, therefore $v \in \{0, 1, -1\}$.



(a) (b)

Fig.2. Examples of faulty network (with abstracted behavioral error models of neurons). (a) Stuck-at-zero faults, where the faulty neurons are missing; (b) Stuck-at-max/min faults, where the faulty neurons are saturated.

TABLE I
DETAILS OF DATASETS AND THEIR SUBNETWORK CONFIGURATION

Name	#Features	#Neurons in each layer	Inference accuracy (fault-free)
MNIST	28×28	784-1024-512-256-256-128	98.87%
Fashion-MNIST	28×28		91.22%
CIFAR-10	32×32	1024-1024-512-256-256-128	71.94%
SVHN	32×32		92.20%

The impact of stuck-at faults in different subnetworks of a TN will be analyzed and evaluated in the next sections. Since this paper targets deep metric learning of the TN, its performance for classification accuracy is considered to assess the impact of faults. Diverse scenarios, including the number and positions of stuck-at faults, are considered. Four popular image classification datasets (MNIST [21], Fashion-MNIST [22], Cifar-10 [23], and SVHN [24]) are used; details of the datasets and their configuration achieving the highest inference accuracy in the fault-free case are provided in Table I.

III. FAULTS IN POSITIVE AND ANCHOR SUBNETWORKS

In this section, the fault tolerance of TNs is evaluated when the stuck-at faults are in the anchor or positive subnetworks. The impact of different factors such as the position of faults, the fault rate and type, is assessed. For a fair comparison, the training in all simulations is set for 20 epochs and the results are averaged over 100 repeated trials.

A. Sensitivity to Layer and Fault Rate

As per Table I, the dimensions of the layers after the input layer in each subnetwork are represented by the vector $[m_1\ m_2\ m_3\ m_4\ m_5] = [1024\ 512\ 256\ 128\ 128]$; stuck-at-max faults are considered in this experiment and the fault rate t is set from 0 to 0.05. Therefore, the fault pattern used in the simulation is $Q_T = (f = \lfloor m_r \times t \rfloor, r \in \{1, \cdots, 5\}, c \in \{anchor\}, v = 1),\ t \in [0, 0.05]$.

This article has been accepted for publication in IEEE Transactions on Emerging Topics in Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TETC.2024.3481962

TABLE II
ACCURACY LOSS WITH 5% OF FAULTY NEURONS IN THE ANCHOR SUBNETWORK

Dataset	Accuracy loss	
	Averaged for all layers	Output layer ($r = 5$)
MNIST	0.0003%	0.0006%
Fashion-MNIST	0.0002%	0.0006%
CIFAR-10	0.0007%	0.0013%
SVHN	0.0004%	0.0009%

Based on the simulation findings, in most cases, the faults lead to a very small degradation of the inference accuracy with the considered fault rate. For example, Table II shows the accuracy loss results for $t = 0.05$ when faults are injected in the anchor subnetwork; both the averaged results for all layers and the worst-case result for the output layer ($r = 5$) are provided. Moreover, a noticeable accuracy loss (> 0.1%) is only found when more than 60% of the neurons are faulty; given that such extensive faults are uncommon in practical scenarios, this paper does not present results for such cases. In addition, there is no discernible difference in cases where faults are injected into different layers of the subnetwork.

From these results, the TN configuration achieves the highest classification accuracy in the fault-free cases (Table I). The output layer is more sensitive to faults; however, the network generally shows a very strong resilience to faults in the anchor subnetwork (also for the positive network as shown in the next subsection). This is expected because the ANNs with a modest size tend to be robust to faults (or noise) due to the non-linear nature and overprovisioning of the network [25]; instead, small faults sometimes even benefit the network by improving generalization [26]. Overall, since the fault rate in practical scenarios is typically low (smaller than 0.05), the presence of stuck-at faults in the anchor/positive subnetworks of a TN only results in a negligible effect that could be compensated by the inherent tolerance during the training process.

B. *Sensitivity to Subnetwork and Fault Type*

The sensitivity of TNs with different types of faults in the anchor and positive subnetworks is presented next. For better illustration, the results are provided on the assumption that the faults are in the output layer with rate $t = 0.05$. Fig. 3 shows the simulated accuracy loss with the pattern $Q_T = (f = m_r \times 0.05, \mathrm{r} \in \{5\}, \mathrm{c} \in \{\mathrm{anchor}, \mathrm{positive}\}, v \in \{0, 1, -1\})$.

The results given in Fig. 3 show that despite both with a very small accuracy loss, the anchor subnetwork exhibits larger sensitivity to faults compared to the positive subnetwork. This is expected because when considering the optimization objective in (2) and (3), the outputs of the anchor subnetwork affect both the positive and negative distances, while the outputs of the positive subnetwork only affect the positive distance.

Moreover, the TN is shown to have better resilience to stuck-at-zero faults than stuck-at-max/min faults when they occur in the positive or anchor subnetwork. This means that hidden units stuck at extreme values usually lead to worse consequences than missing hidden units or connection failure. Therefore, we have studied the distribution of neuron values (after the activation function tanh) during the training process. Due to the data normalization, the distribution is approximately Gaussian with more than 92% of values concentrated in the range $[-0.1, 0.1]$. Hence, a possible explanation is that being stuck at zero is closer to the correct values of the fault-free cases, which leads to a smaller degradation than being stuck at the extreme values (which are rather uncommon during normal operation).

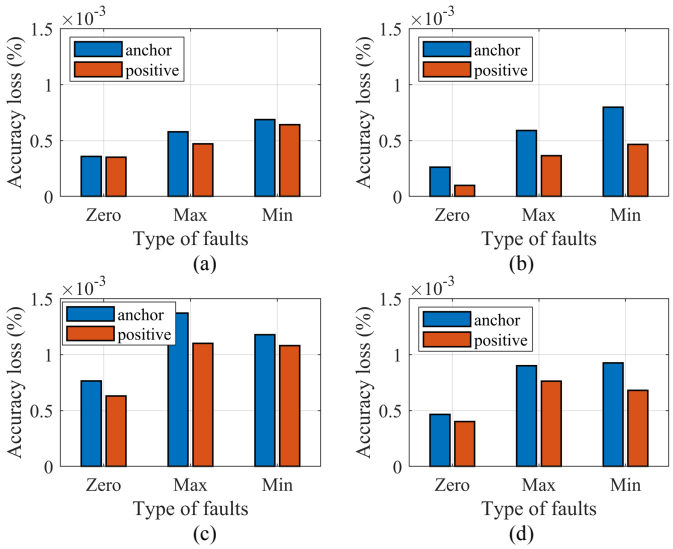


Fig. 3. Accuracy loss caused by different types of faults in the anchor and positive subnetworks for dataset (a) "MNIST"; (b) "Fashion-MNIST"; (c) "CIFAR-10"; (d) "SVHN". "Zero", "Max" and "Min" denote the stuck-at-zero, stuck-at-max and stuck-at-min faults, respectively.

IV. FAULTS IN NEGATIVE SUBNETWORK

Different from the case for positive and anchor subnetworks, stuck-at faults in the negative subnetwork of a TN can lead to a failure in training. This occurs due to the so-called "false solution" caused by stuck-at faults, which results in an incorrect convergence of training. In this section, the issue of training failure in the presence of stuck-at faults in the negative subnetwork is initially presented. Then, the occurrence of a false convergence due to faults is analyzed. Finally, the impact of faults with different influence factors on the network performance is further evaluated.

A. *Training Failure*

As per the simulations, the training of TNs can generate the following three possible convergence results when stuck-at faults are injected into the negative subnetwork.

- ***Failure-free:*** The convergence of the TN training is not (or almost not) affected by faults. The network performs the same (or very close) behavior as the fault-free case.
- ***Partial failure:*** The TN initially performs a normal convergence as the *failure-free* case, while it suddenly converges to the false solution after several training epochs. In this scenario, the final inference accuracy will fall between the *failure-free* and *complete failure* cases, depending on the training progress prior to converging to a false solution.
- ***Complete failure:*** The TN converges to the false solution at the beginning of the training process. In this scenario, the network stops updating at a very low inference accuracy, such as around 10%; this suggests that the model completely fails for the classification

This article has been accepted for publication in IEEE Transactions on Emerging Topics in Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TETC.2024.3481962

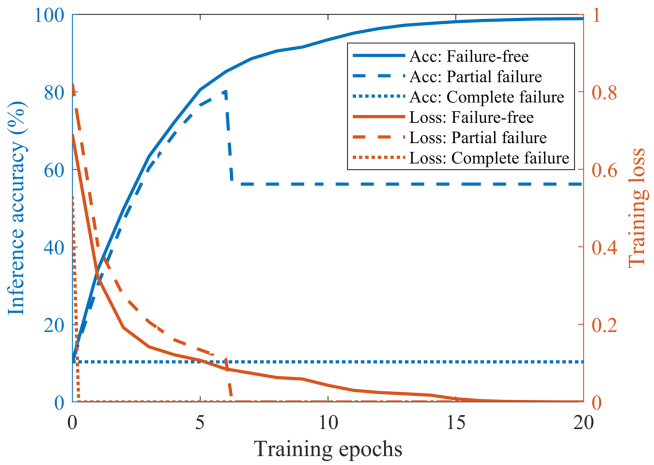


Fig. 4. Example of the change in inference accuracy and training loss with training epochs for dataset "MNIST", when different convergence cases (*failure-free*, *partial failure* and *complete failure*) occur.

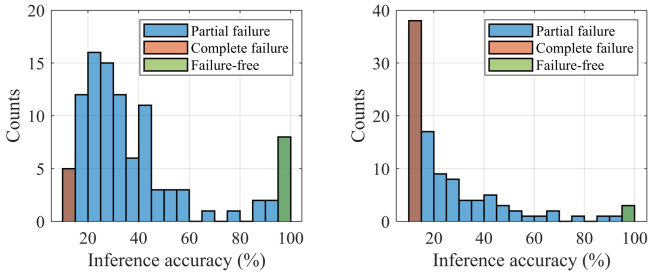


Fig. 5. Distributions of inference accuracy with 100 repeated trials for dataset "MNIST" under stuck-at-max faults with: (a) $t = 0.02, r = 1$ (average accuracy: 39.95%); (b) $t = 0.02, r = 2$ (average accuracy: 24.65%).

task (samples are arbitrarily predicted as one of the pre-known classes).

Fig. 4 shows an example to illustrate these three cases by injecting faults to the negative subnetwork of the TN for dataset "MNIST"; the change of the training loss and the inference accuracy with training epochs are plotted in this figure. In the *failure-free* case, the inference accuracy keeps increasing while the training loss decreases with the iterative propagations. For the case of *complete failure*, the training loss directly reduces to 0 once the training starts, and the network remains untrained with a very low inference accuracy. For the case of *partial failure*, the training loss is normal at the beginning but then, it suddenly reduces to 0; while the inference accuracy also shows a rapid drop and remains at a low value after the network reaches a false convergence.

All these convergence cases can occur even with a fixed network configuration and fault injection method; this is due to randomization in training, such as weight initialization and batch arrangement. Fig. 5 shows the distribution of inference accuracy (associated with different convergence cases) with 100 repeated trials using the dataset "MNIST". In these figures, the first bin (for an accuracy of 10% to 15%) can be approximately considered for the *complete failure* case; the last bin (for an accuracy of 95% to 100%) represents the *fault-free* case, and the remaining bins represent the *partial failure* cases. These results show that both *partial failure* and *complete failure* cases are very likely to occur. As explained at the beginning of this section, these failures are generated by the false solution of training when the negative subnetwork is affected by faults; such impact is analyzed in the next subsection.

B. *False Convergence*

1) *Faults in the output layer*

The issue of deceptive convergence caused by stuck-at-zero faults in the output layer of the negative subnetwork ($r = l, v = 0$) is considered first. The incorrect convergence caused by such faults is rather intuitive. Let I be the set of indices of the stuck-at-zero faults and i denote the index of the faulty neurons ($i \in I$). When the stuck-at-zero faults are injected, we have $y_{n,i}^l = v_i$; as per (3), the constraint of the negative distance in this case is given by

$$d_- = \sqrt{\sum_{\substack{i=1,\\ i\notin I}}^{m_l} \left(y_{a,i}^l - y_{n,i}^l\right)^2 + \sum_{i\in I}\left(y_{a,i}^l - v_i\right)^2} \geq \sum_{i\in I}\left|y_{a,i}^l - v_i\right| > \varepsilon_- \quad (5)$$

In (5), the anchor output $y_{a,i}^l$ can reach an extreme value after the activation computation, such as 1 or -1 for tanh; this leads to the maximum impact of the faulty terms as $v_i = 0$. When calculating the loss in (4) during training, the false solution to achieve zero loss is easily achieved with the commonly used margin $M = 1$. This scenario is illustrated in Fig. 6. Therefore, when the negative subnetwork has stuck-at-zero faults, the false convergence is very likely to occur, because it only needs to map the anchor and positive subnetworks to similar outputs to meet (2), while making $y_{a,i}^l$ large enough to make use of the faulty term in the negative subnetwork. Once the faulty term dominates d_-, the weight update stops with $L_{TL} = 0$.

Overall, the false convergence is more likely to occur in very few epochs, causing *partial failure*. It may also occur at the beginning of training (when $d_+ \approx d_-$), causing *complete*

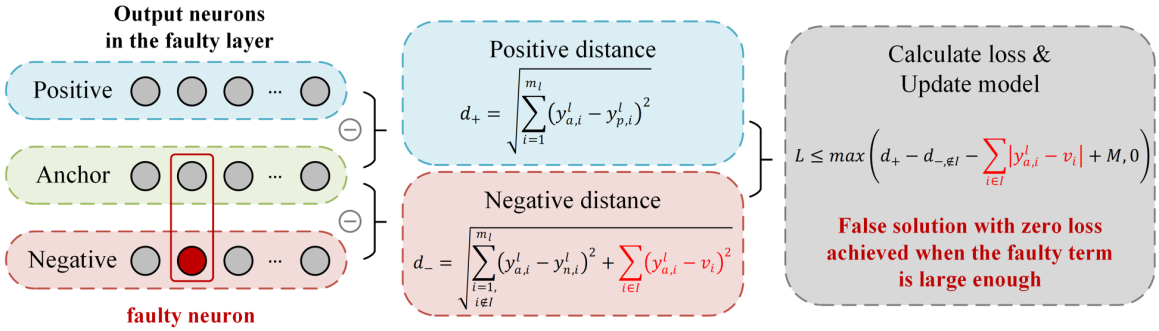


Fig. 6. Training of TNs with the false solution caused by the stuck-at faults in the negative subnetwork. The red parts in the equations denote the faulty terms.

This article has been accepted for publication in IEEE Transactions on Emerging Topics in Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TETC.2024.3481962

failure, because a single faulty term $\left|y_{a,i}^{l}-v_i\right|$ in d_- (can be up to 1) in the first epoch almost satisfies the constraint of (4).

Next, let I be the set of indices of the stuck-at-max/min faults; the analysis of false convergence is similar to the case of stuck-at-zero faults. Consider (5) for $v_i=1$ or -1; the constraint of d_- can also be satisfied by only considering the faulty terms $\left|y_{a,i}^{l}-v_i\right|$. The analysis is like the previous subsection, and differently, the faulty term $\left|y_{a,i}^{l}-v_i\right|$ can be up to 2 (instead of 1) if we take $y_{a,i}^{l}$ as the opposite extreme value of v_i.

A dominant faulty term, potentially with a larger absolute value, can exacerbate the situation. This implies that the network is even more susceptible to false convergence with stuck-at-max/min faults. In this case, a common situation is that the convergence can get stuck into the zero-loss situation even before the training starts. For all loss functions optimizing the relaxed constraint on d_+-d_-, this leads to false convergence from the start. For example, the L_{TL} in (4) can be represented as

$$L_{TL}=max(d_+-d_-+M,0)$$

$$=max\left(d_+-\sqrt{\sum_{\substack{i=1,\\ i\notin I}}^{m_l}\left(y_{a,i}^{l}-y_{n,i}^{l}\right)^2+\sum_{i\in I}\left(y_{a,i}^{l}-v_i\right)^2}+M,0\right)$$

$$\le max\left(d_+-d_{-,\notin I}-\sum_{i\in I}\left|y_{a,i}^{l}-v_i\right|+M,0\right) \quad (6)$$

where $v_i=1$ or -1, and M is commonly equal to 1. The faulty term leads to a false solution with zero-loss if $\sum_{i\in I}\left|y_{a,i}^{l}-v_i\right|\ge d_+-d_{-,\notin I}+M$. $d_+-d_{-,\notin I}$ is usually much smaller than 1 with an appropriate random initialization of weights; in this case, even when considering only one faulty term with extreme value, $L_{TL}=0$ can be achieved. Therefore, the training process is very likely to be trapped with false convergence at the very beginning, causing *complete failure*. Note that all types of faults can lead to *partial failure* or *complete failure*; however, the stuck-at-zero (max/min) faults faults are more likely to result in *partial* (*complete*) *failure*.

2) Faults in hidden layers

The previous analysis on false convergence due to faults in the output layer can be similarly extended to cases when the faults are in the hidden layers (i.e., $r<l$); however, when considering the complexity of the forward propagation, a strict representation of constraints in each layer cannot be established. Next, we provide a heuristic discussion. The propagation between fully-connected layers is given by

$$y_j^{r+1}=\phi\left(\sum_i w_{j,i}^{r}\cdot y_i^{r}+b_j^{r}\right) \quad (7)$$

where $w_{j,i}^{r}$ and b_j^{r} are the weight and bias; ϕ is the activation function ("tanh" used in this paper).

Similar to the constraints at the output layer, the faulty terms $\sum_{i\in I}\left(y_{a,i}^{r}-v_i\right)$ can still dominate at the r-th layer. Such distances in the negative subnetwork can spread to the following layers (so, no longer concentrated on those faulty indices, but separate to all subsequent units). However, $tanh(x)$ bounds layer outputs to the range $[-1,1]$, and weakens the dominance of the large distances in the faulty layer. This behavior accumulates across multiple layers; consequently, when faults manifest in the early layers, their final impact on the negative distance d_- at the output layer tends to be less pronounced. In such cases, the TN typically requires a longer time to converge to a false solution, increasing the likelihood of reaching a *partial failure*.

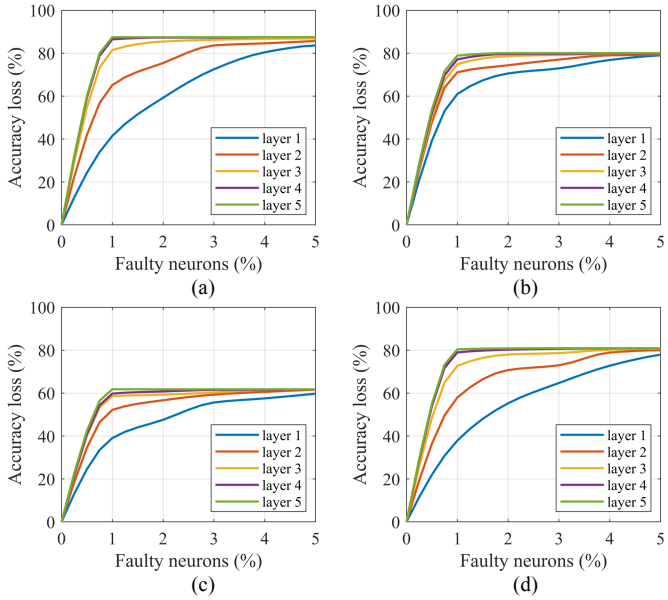


Fig. 7. Accuracy loss caused by stuck-at-max faults in the negative subnetwork with different positions and rates for dataset: (a) "MNIST"; (b) "Fashion-MNIST"; (c) "CIFAR-10"; (d) "SVHN".

Furthermore, it is evident that an increasing number of faulty neurons will amplify their effects and the TN is more prone to get stuck at the false solution, leading to a *complete failure*. This can be indicated in (5) or (6) by more faulty terms. As important influence factors, the position of the faulty layers and the rate of faulty neurons will be further evaluated by fault injection experiments in the following subsection.

It is worth mentioning that even though this paper takes the most popular TL (4) as an example, the above analysis is equally applied to other types of loss functions in TNs. This is applicable because most of the loss functions aim to decrease the "difference" term [27], i.e., d_+-d_-. For example, the loss functions introduced in [1], [28], [29] can be represented as $L=g(d_+-d_-)$, where $g(\cdot)$ is the function specified in different literature. Even though for some unconventional loss functions that are independent of the "difference" term [27], they cannot remove the constraints in (2) and (3), because they are the fundamental objectives of the TNs. Therefore, the analysis presented in Section IV.B can be extended to TNs with all types of loss functions.

C. Sensitivity to Layer, Fault Rate and Fault Type

This subsection evaluates the impact of false convergence on the TN performance, when different numbers and types of faults are injected into different layers of the negative network. Like the evaluation for faults in the positive/anchor subnetwork presented in Section III, the rate of faulty neurons t and the position of faulty layers r, are considered first.

Fig. 7 shows the simulation results with the fault pattern $Q_T=(f=m_r\times t, r\in\{1,\cdots,5\}, c\in\{negative\}, v\in\{1\})$, $t\in[0,0.05]$. As shown in Fig. 7, a significant accuracy loss is caused by only a very low rate of faulty neurons in the negative subnetwork. This is different from the high resilience of the anchor and positive subnetworks evaluated in Section III-A.

This article has been accepted for publication in IEEE Transactions on Emerging Topics in Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TETC.2024.3481962

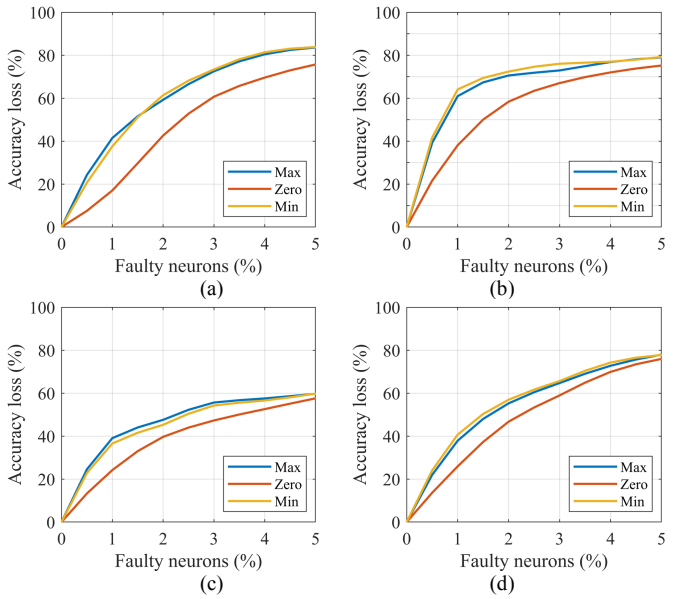


Fig. 8. Accuracy loss caused by faults in the negative subnetwork with different rates and types for dataset: (a) "MNIST"; (b) "Fashion-MNIST"; (c) "CIFAR-10"; (d) "SVHN". "Zero", "Max" and "Min" denote the stuck-at-zero, stuck-at-max and stuck-at-min faults, respectively.

Moreover, for all datasets, the accuracy loss reaches its peak level showing that all repeated trials reach the false solution. These results verify the analysis presented in the previous subsection, which indicates that faults in the negative subnetwork can simply lead to false convergence. Also, the accuracy loss tends to be larger when faults affect the latter layers; this is also in line with the previous analysis: these layers are more likely to result in *complete failure*, consequently leading to a substantial loss in accuracy.

The second experiment is conducted to evaluate the impact of different fault types on the network performance. Different types of stuck-at faults with different rates are injected into layer $r = 1$. Fig. 8 shows the simulation results with the pattern $Q_T = (f = m_r \times t, r \in \{1\}, c \in \{negative\}, v \in \{0,1,-1\})$, $t \in [0, 0.05]$. The results indicate that stuck-at-max/min faults generally result in a greater loss of accuracy compared to stuck-at-zero faults faults. This observation aligns with the previous analysis; the neurons stuck at extreme values exhibit larger faulty terms, leading to more significant degradation. This behavior is also shown to be independent of the fault rate as per Fig. 8.

V. Proposed Fault-tolerant Schemes

The analysis and simulations presented in this paper indicate the necessity of fault-tolerant strategies to address stuck-at faults within the TNs' negative subnetworks. Consequently, we present a scheme containing two parts that handle stuck-at-zero and stuck-at-max/min faults respectively; the combined use of these two parts is essentical to prevents TNs from all types of stuck-at faults. Simulations have been run to confirm the effectiveness of the proposed approach.

A. Regularization to Anchor Outputs

From the analysis in Section IV.B, the false convergence caused by stuck-at-zero faults occurs when the outputs of the anchor and positive subnetworks ($y_{a,i}^l$ and $y_{p,i}^l$) have large absolute values. In this case, the constraint of (5) can be simply satisfied by the faulty term. Therefore, the first proposed fault-tolerant scheme aims to prevent an incorrect convergence towards large outputs in the anchor or positive terms.

To achieve this goal, an extra penalty term (i.e., the regularization term) on the anchor outputs from different layers is proposed to be added to the loss function; so, (4) is updated as

$$L_{proposed} = max(d_+ - d_- + M, 0) + \lambda \sum_{j=1}^{l} \sum_{i=1}^{m_j} {y_{a,i}^j}^2 \leq \varepsilon \quad (7)$$

where m_j is the dimension of the layer j and λ is the parameter determining the influence of the regularization term. This regularization term can be involved in either the anchor subnetwork or the positive subnetwork, because their outputs tend to be the same due to the objective function. In our case using the traditional loss function, the regularized anchor outputs affect the calculation of both positive and negative distances, so it is performed to better accelerate convergence.

In general, regularization is added to all layers of the subnetwork, but when the faulty layers can be specified, it is only required for those outputs. However, incorporating a regularization term for all potential faulty layers can still be expensive in propagations, particularly in the case of large-scale ANNs. A practical approach is to apply a filter to the outputs (prior to activation functions) for detection purposes; its principle and implementation are like the weight filter introduced in [30]. This is based on the observation that the outputs (before the non-linear mapping by the activation functions) of fault-free layers are small (usually smaller than 1); however, when the faulty term dominates, the absolute values of the corresponding outputs can become more than 100 times larger. Therefore, once the filter detects outliners, the penalty term in (7) can be set to only regularize the specified layers. The threshold of the filter can be set as an empirical maximum/minimum of the layer outputs as in [30]. By using this approach, computational costs can be reduced without the need to regularize all potential faulty layers.

In the proposed scheme, the regularization term on the anchor outputs can restrict the faulty units from taking a large value; by choosing a proper parameter λ, the regularization term can prevent the false convergence that causes *partial failures*. However, it does not eliminate *complete failures* caused by stuck-at-max/min faults. Therefore, another protective method is required as presented in the following subsection.

B. Modified Margin

The *complete failure* is likely to happen when the fault rate is large, especially for stuck-at-max/min faults that potentially lead to larger faulty terms in (6). In this scenario, the TN can be trapped at the false convergence at the beginning of the training process, even before conducting the regularization term functions. Therefore, an intuitive way to address this issue is to increase the original margin M to avoid the loss getting stuck at the false convergence very soon.

For each faulty term $|y_{a,i}^l - v_i|$, the maximum error is 2 when $v_i = 1$ or $v_i = -1$, $-1 \leq y_{a,i}^l \leq 1$. By (6), if the number of faulty neurons is f, increasing the margin by $2f$ can ensure that the convergence does not achieve a false solution over the

TABLE III
ACCURACY LOSS WITH 5% OF FAULTY NEURONS IN THE NEGATIVE SUBNETWORK USING THE PROPOSED SCHEMES

Dataset	Averaged accuracy loss		
	stuck-at-zero	stuck-at-max	stuck-at-min
MNIST	0.00027%	0.00047%	0.00053%
Fashion-MNIST	0.00021%	0.00042%	0.00048%
CIFAR-10	0.00060%	0.00108%	0.00093%
SVHN	0.00042%	0.00075%	0.00067%

entire training process (as the initial outputs $y_{a,i}^{l}$ are usually close to 0). However, the number of faulty neurons is usually unknown in many practical applications. An appropriate margin is critical to address the importance of the contributive triplets [27]; setting the margin to an arbitrarily large value slows down the convergence in the fault-free case. Therefore, increasing the margin must be thought carefully.

We propose to modify the margin at the beginning of the training by:

$$M' = M + 2\lfloor d\_ \rfloor \tag{8}$$

where $\lfloor d\_ \rfloor$ denotes the round-down to the largest integer smaller than $d\_$. Note that the margin only changes based on the output at the first iteration, and it is fixed in the following training process. This value assumes that the initial negative distance is smaller than 1 in the fault-free case (valid for all applied datasets); thus, it can be an estimate of the number of stuck-at-max/min faults. Consider the widely used process of weight initialization; this estimate works for most cases, and it can also be modified by the empirical knowledge of different applications. In summary, the modified margin M' is required to correctly start the training process and prevent *complete failures* when stuck-at faults are present in the negative subnetwork.

C. Evaluation

This subsection provides the simulation results for the proposed fault-tolerant methods. Both the regularization to the anchor outputs and the modified margin are required, as the false convergences lead to both *partial* and *complete failures* as illustrated in Fig. 5. In practice, the modified margin converts the cases of *complete failures* to *partial failures*, and then the regularization eliminates the false convergence during the training process. Since each of them could only conditionally solve the failures and the component of failure types is not predictable a-priori, the two steps should be applied simultaneously in the proposed scheme.

The fault injection process and the network configurations are the same as those applied in the previous sections. Simulation has been performed for faults injected into the output layer, so also shows the worst case. The pattern of the fault injection simulation is given by $Q_T = (f = m_r \times t\,, r \in \{5\}, c \in \{negative\}, v \in \{0,1,-1\})$, $t = 0.05$.

Table III reports the accuracy loss of the TN protected by the proposed schemes with the given fault pattern; results are averaged among 100 trails. Compared with the unprotected results provided in Figs. 7 and 8, the accuracy loss significantly decreases, and it is smaller than 0.00108%; moreover, it is low also when the faults are in the positive/anchor subnetworks (Fig. 3). These results indicate that a TN can almost fully tolerate the stuck-at faults by using the proposed schemes. Therefore, in practical applications, the proposed fault-tolerant methods can be utilized in TNs to avoid training failures caused by stuck-at faults.

VI. LIMITATIONS

The validity and applicability of the findings and the proposed protection techniques depend on the assumed model for faults as well as errors; in this paper, this aspect was discussed in Section II. The analysis presented in this manuscript is defined at a functional level, so independent of the implementation. Therefore, for a given implementation, the fault model can be different and thus, the validity of the results and protection techniques here presented may not be fully applicable. As the proposed schemes and analysis are generic, they can provide a starting point to study the dependability of TNs (as well as other NNs) in specific schemes and applications. The generality of the results of this manuscript capture the basic phenomena and provide protection schemes that can be also useful to guide the design under specific conditions, such as physical based fault models.

VII. CONCLUSION

This paper has comprehensively studied the fault-tolerance of Triplet Networks (TNs) when stuck-at faults are present during the training process. Based on the analysis and simulation results, the anchor and positive subnetworks of the TN are shown to be insensitive to stuck-at faults; however, faults in the negative subnetwork can cause false convergence, leading to *partial failure* and *complete failure*. In addition to the analysis on the occurrence of false convergence, the impact of faults with different positions, rates, and types on the network performance have also been evaluated. It has been shown that a larger fault rate leads to greater accuracy loss, until all trials fall into the *complete failure* case; moreover, the TN is more sensitive to faults in the layers closer to the output layer as well as the stuck-at-max/min faults.

Furthermore, two fault-tolerant methods have been proposed to handle the faults in the negative subnetworks. Specifically, the loss function with regularization on the anchor outputs has been proposed to avoid *partial failures*; the so-called modified margin with improper initializations has been proposed to avoid *complete failures*. Simulation results have shown that by applying both approaches simultaneously, the accuracy loss of the TN significantly decreases compared to the unprotected scheme and it is nearly zero (up to 0.00108%); therefore, the proposed methods can be used for TNs to avoid the false convergence and training failures caused by stuck-at faults. Since this paper focuses on the impact of high-level functional characterization of the stuck-at fault model on TNs, a more detailed validation of the low-level fault model that is related to specified implementations, will be further investigated in future works.

REFERENCES

[1] E. Hoffer, and N. Ailon, "Deep metric learning using triplet network," *International workshop on similarity-based pattern recognition*. Springer, Cham, pp. 84-92, Nov. 2015.

[2] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering." *Proceedings of the IEEE*

conference on computer vision and pattern recognition, pp. 815-823, 2015.

[3] Y. Zhang, D. Liu, and Z. J. Zha, "Improving triplet-wise training of convolutional neural network for vehicle re-identification," *2017 IEEE International Conference on Multimedia and Expo (ICME)*, IEEE, pp. 1386-1391, Jul. 2017.

[4] H. Lai, J. Chen, L. Geng, Y. Pan, X. Liang and J. Yin, "Improving Deep Binary Embedding Networks by Order-Aware Reweighting of Triplets," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 4, pp. 1162-1172, April 2020.

[5] V. Kumar BG, G. Carneiro, and I. Reid. "Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions." *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5385-5394, 2016.

[6] J. A. Abraham and W. K. Fuchs, ``Fault and error models for VLSI," *Proc. IEEE*, vol. 74, no. 5, pp. 639-654, May 1986.

[7] C. Torres-Huitzil and B. Girau, "Fault-tolerance in neural networks: Neural design and hardware implementation," *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1-6, Dec. 2017.

[8] L. Matanaluza *et al*., "Emulating the Effects of Radiation-Induced Soft-Errors for the Reliability Assessment of Neural Networks," in *IEEE Transactions on Emerging Topics in Computing*.

[9] S. Liu, K. Chen, P. Reviriego, W. Liu, A. LOURI and F. Lombardi, "Reduced Precision Redundancy for Reliable Processing of Data," in *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1960-1971, 1 Oct.-Dec. 2021.

[10] V. Piuri, "Analysis of fault tolerance in artificial neural networks," *Journal of Parallel and Distributed Computing,* vol. 61, no.1, pp. 18-48, 2001.

[11] T. Haruhiko, M. Masahiko, K. Hidehiko and H. Terumine, "Enhancing both generalization and fault-tolerance of multilayer neural networks," *2007 International Joint Conference on Neural Networks*, pp. 1429-1433, Aug. 2007.

[12] N. Kamiura, Y. Taniguchi, T. Isokawa and N. Matsui, "An improvement in weight-fault-tolerance of feedforward neural networks," *Proceedings 10th Asian Test Symposium*, pp. 359-364, Aug. 2001.

[13] B. S. Arad and A. El-Amawy, "On fault-tolerant training of feedforward neural networks", *Neural Netw.*, vol. 10, no. 3, pp. 539-553, 1997.

[14] C. H. Sequin and R. D. Clay, "Fault-tolerance in artificial neural networks", *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 1, pp. 703-708, Jun. 1990.

[15] S. Ding, *et al.,* "Deep feature learning with relative distance comparison for person re-identification," *Pattern Recognition*, vol 48, no. 10, pp. 2993-3003, Oct. 2015.

[16] C. Torres-Huitzil and B. Girau, "Fault and Error Tolerance in Neural Networks: A Review," in *IEEE Access*, vol. 5, pp. 17322-17341, Aug. 2017.

[17] R. A. Nawrocki and R. M. Voyles, "Artificial neural network performance degradation under network damage: Stuck-at faults," *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, USA, pp. 442-449, 2011.

[18] B. Salami, O. S. Unsal and A. C. Kestelman, "On the Resilience of RTL NN Accelerators: Fault Characterization and Mitigation," *2018 30th International Symposium on Computer Architecture and High Performance Computing*, Lyon, France, pp. 322-329, 2018.

[19] L. Xia et al., "Stuck-at Fault Tolerance in RRAM Computing Systems," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 102-115, 2018.

[20] Y. Liu and C. Huang, "Scene Classification via Triplet Networks," in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 1, pp. 220-237, Jan. 2018.

[21] Y. LeCun, L. Bottou, Y. Bengio et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[22] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," in *arXiv preprint arXiv:1708.07747*, 2017.

[23] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, 2009.

[24] Y. Netzer, T. Wang, A. Coates, et al., "Reading digits in natural images with unsupervised feature learning," 2011.

[25] G. An, "The effects of adding noise during backpropagation training on a generalization performance, " *Neural computation*, vol. 8, no. 3, pp.643-674, Apr.1996.

[26] C. M. Bishop, "Training with Noise is Equivalent to Tikhonov Regularization," in *Neural Computation*, vol. 7, no. 1, pp. 108-116, Jan. 1995.

[27] Y. Liu and C. Huang, "Scene Classification via Triplet Networks," in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 1, pp. 220-237, Jan. 2018.

[28] D. Cheng, *et al.*, "Person re-identification by multi-channel parts-based cnn with improved triplet loss function," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[29] J. Yu, C. Zhu, J. Zhang, Q. Huang and D. Tao, "Spatial Pyramid-Enhanced NetVLAD With Weighted Triplet Loss for Place Recognition," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 2, pp. 661-674, Feb. 2020.

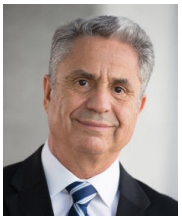
[30] Z. Wang, F. Niknia, S. Liu, P. Reviriego, P. Montuschi and F. Lombardi, "Tolerance of Siamese Networks (SNs) to Memory Errors: Analysis and Design," in *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 1136-1149, 1 April 2023.

Ziheng Wang (S'21) received the BEng degree in electronic and information engineering from Harbin Institute of Technology, Harbin, China, in 2018, and the MS degree in electrical engineering from University of Pennsylvania in 2020. He is pursuing the PhD degree as a research assistant in the Department of Electrical and Computer Engineering, Northeastern University. His current research direction is high-prformance neural networks including fault tolerance and stochastic computing.

Farzad Niknia (S'21) received the B.Sc. and M.Sc. degrees in Electrical and Electronics Engineering from the University of Tabriz, Iran in 2014 and 2018. He worked as a research assistant at IC Design Lab through his M.Sc. degree. He is currently working towards the Ph.D. degree in Computer Engineering at Northeastern University, Boston as a research assistant with a concentration on ASIC Design. His research interests include ASIC and FPGA design, VLSI, EDA tools, design for test and hardware security.

Shanshan Liu (M'19-SM'23) received the Ph.D. degree in Microelectronics and Solid-State Electronics from Harbin Institute of Technology, Harbin, China, in 2018. She was a postdoc researcher with Northeastern University, Boston, USA, from 2018 to 2021, and an assistant professor with New Mexico State University, Las Cruces, from 2021 to 2023. She is currently a professor with University of Electronic Science and Technology of China, Chengdu, China. She serves as an Associate Editor for the IEEE Transactions on Emerging Topics in Computing and the IEEE Transactions on Nanotechnology, and a Guest Editor for the IEEE Transactions on Circuits and Systems-I. Her research interests include fault-tolerance design in high performance computing systems, emerging computing, VLSI design, dependable machine learning, error correction codes.

Pedro Reviriego (M'04-SM'15) received the M.Sc. and Ph.D. degrees in telecommunications engineering from the Technical University of Madrid, Madrid, Spain, in 1994 and 1997, respectively. From 1997 to 2000, he was an Engineer with Teldat, Madrid, working on router implementation. In 2000, he joined Massana to work on the development of 1000BASE-T transceivers. From 2004 to 2007, he was a Distinguished Member of Technical Staff with the LSI Corporation, working on the development of Ethernet transceivers. From 2007 to 2018 he was with Nebrija University and from 2018 to 2022 with Universidad Carlos III de Madrid. He is currently with Universidad Politécnica de Madrid working on several topics in computer science with a focus on security, privacy and reliability. He is currently Associate editor of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING.



Ahmed Louri is the David and Marilyn Karlgaard Endowed Chair Professor of Electrical and Computer Engineering (ECE) at the George Washington University, which he joined in August 2015. He received the Ph.D. degree in Computer Engineering from the University of Southern California, Los Angeles, California in 1988. From 1988 to 2015, he was a professor of ECE at the University of Arizona. From 2010 to 2013, he served as a program director in the National Science Foundation's Directorate for Computer and Information Science and Engineering. His research interests are interconnection networks and network on chips for multicores, and the use of machine learning techniques for energy-efficient, reliable, high-performance and secure many-core architectures and accelerators. He was selected to be the recipient of the IEEE Computer Society 2020 Edward J. McCluskey Technical Achievement Award. He is currently serving as the Editor-in-Chief of IEEE TRANSACTIONS ON COMPUTERS.

Fabrizio Lombardi (M'81-SM'02-F'09) received the B.Sc. degree (Hons.) in electronic engineering from the University of Essex, U.K., in 1977, the master's degree in microwaves and modern optics and the Diploma degree in microwave engineering from the Microwave Research Unit, University College London, in 1978, and the Ph.D. degree from the University of London in 1982. He is currently the International Test Conference (ITC) Endowed Chair Professorship with Northeastern University, Boston, USA. His research interests are bio-inspired and nano manufacturing/computing, VLSI design, testing, and fault/defect tolerance of digital systems. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTERS from 2007 to 2010 and the inaugural Editor-in-Chief of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING from 2013 to 2017. He is currently the Vice President for Publications of the IEEE Computer Society and a member of the IEEE Publication Services and Products Board.