Fault Tolerant Triplet Networks for Training and Inference

Ziheng Wang [1], Farzad Niknia [1], Shanshan Liu [2], Pedro Reviriego [1], Ahmed Louri [1], and Fabrizio Lombardi [1]

[1]Affiliation not available
[2]New Mexico State University

October 05, 2022

Abstract

This paper deals with the fault tolerance of Triplet Networks (TNs). Results based on extensive analysis and simulation by fault injection are presented for new schemes. As in accordance with technical literature, stuck-at faults are considered in the fault model for the training process. Simulation by fault injection shows that the TNs are not sensitive to this type of fault in the general case; however, an unexcepted failure (leading to network convergence to false solutions) can occur when the faults are in the negative subnetwork. Analysis for this specific case is provided and remedial solutions are proposed (namely the use of the loss function with regularized anchor outputs for stuck-at 0 faults and a modified margin for stuck-at 1/-1 faults). Simulation proves that false solutions can be very efficiently avoided by utilizing the proposed techniques. Random bit-flip faults are then considered in the fault model for the inference process. This paper analyzes the error caused by bit-flips on different bit positions in a TN with Floating-Point (FP) format and compares it with a fault- tolerant Stochastic Computing (SC) implementation. Analysis and simulation of the TNs confirm that the main degradation is caused by bit-flips on the exponent bits. Therefore, protection schemes are proposed to handle those errors; they replace least significant bits of the FP numbers with parity bits for both single- and multi-bit errors. The proposed methods achieve superior performance compared to other low-cost fault tolerant schemes found in the technical literature by reducing the classification accuracy loss of TNs by 96.76% (97.74%) for single-bit (multi-bit errors).

Fault Tolerant Triplet Networks for Training and Inference

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

***Abstract*—This paper deals with the fault tolerance of Triplet Networks (TNs). Results based on extensive analysis and simulation by fault injection are presented for new schemes. As in accordance with technical literature, stuck-at faults are considered in the fault model for the training process. Simulation by fault injection shows that the TNs are not sensitive to this type of fault in the general case; however, an unexcepted failure (leading to network convergence to false solutions) can occur when the faults are in the negative subnetwork. Analysis for this specific case is provided and remedial solutions are proposed (namely the use of the loss function with regularized anchor outputs for stuck-at 0 faults and a modified margin for stuck-at 1/-1 faults). Simulation proves that false solutions can be very efficiently avoided by utilizing the proposed techniques. Random bit-flip faults are then considered in the fault model for the inference process. This paper analyzes the error caused by bit-flips on different bit positions in a TN with Floating-Point (FP) format and compares it with a fault-tolerant Stochastic Computing (SC) implementation. Analysis and simulation of the TNs confirm that the main degradation is caused by bit-flips on the exponent bits. Therefore, protection schemes are proposed to handle those errors; they replace least significant bits of the FP numbers with parity bits for both single- and multi-bit errors. The proposed methods achieve superior performance compared to other low-cost fault tolerant schemes found in the technical literature by reducing the classification accuracy loss of TNs by 96.76% (97.74%) for single-bit (multi-bit errors).**

***Index Terms*— Triplet Network, fault tolerance, stuck-at faults, random bit-flips.**

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 their 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 safety-critical applications, fault tolerance is important for hardware implementation of ANNs, because faults occurred in the hardware and their erroneous consequences may lead to performance degradation and sometimes to a total system failure [6], [7], [8]. Traditional fault-tolerant methods usually require redundancy [9] and schemes for reliable ANNs mainly include: i) introducing redundancy of critical neurons or branches [10], [11], ii) changing backpropagation algorithms, such as adding penalty terms or modifying the weights [12], and iii) training the network by artificially injected faults [13]. However, the focus of these methods is often on networks of small size and simple structure; the fault tolerance of large-scale applications such as TNs has not been sufficiently explored. Since the three-subnetwork structure of TNs and their capability of similarity-measuring based on limited training data may lead to unique problems in the presence of faults, it is of paramount importance to investigate the fault tolerance of TNs. To the best of the authors' knowledge, this has not been reported in the technical literature.

This paper focuses on the application of deep metric learning classification, whose subnetwork is usually simpler for fault injection experiments and theoretical analysis; by sharing a similar structure, the methods/conclusions are expected to extend to further tasks with TNs. The fault tolerance of TNs implemented by MLPs or CNNs as subnetworks are comprehensively studied. Two widely used fault models, stuck-at faults and random bit-flips, are considered for the training and inference processes respectively (as also found in [13] and [14]); this is based on the consideration that the bit-flips are not critical in training as transient faults, while they have a much more significant effect during inference as shown in simulations. In addition to the analysis and evaluation of the impact of faults on the classification performance of TNs, efficient protection solutions for both the training and inference processes are also proposed.

The main contributions of this paper are concisely described as follows:

1) By analyzing the fault models in the training process, a special case of incorrect convergence has been revealed when stuck-at faults are injected into the negative

Ziheng Wang, Farzad Niknia and Fabrizio Lombardi are with Department of Electrical and Computer Engineering, Northeastern University, MA 02115, USA (email: wang.zihe@northeastern.edu, niknia.f@northeastern.edu, lombardi@ece.neu.edu).

Shanshan Liu is with Klipsch School of Electrical and Computer Engineering, New Mexico State University, NM 88001, USA (email: ssliu@nmsu.edu).

Pedro Reviriego is with the Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, 28040 Madrid, Spain (email: pedro.reviriego@upm.es).

Ahmed Louri is with the Department of Electrical and Computer Engineering, George Washington University, DC 20052, USA (email: louri@gwu.edu).

subnetwork of the TN, which can lead to complete system failures.

2) The impact of stuck-at faults in the training process has been evaluated by simulations. The special case in the negative subnetwork has been also verified.
3) Solutions have been proposed for the incorrect convergence due to a faulty negative subnetwork. The effectiveness of the proposed methods has been shown by simulations.
4) Error analysis has been pursued for the random bit-flips in the inference process. The feature of floating-point (FP) and Stochastic Computing (SC) has also been considered and compared.
5) Simulation has been used to assess the fault tolerance for bit-flips in the inference process. Data represented by floating-point (FP) format suffer an accuracy loss due to extremely large outliers caused by faults on the most significant bits.
6) Protection schemes based on parity bits are proposed for both single-bit and multi-bit errors. Simulation has shown that the proposed methods achieve excellent performance compared with other low-cost fault tolerance approaches.

The rest of the paper is organized as follows. Section II outlines preliminaries, such as the structure of TNs and SC. Section III introduces the stuck-at faults as the fault model for training, and the special case with faults in the negative channel is analyzed in detail in Section IV. Section V evaluates the impact of faults on training performance for both general and special cases. As solutions to mitigate such impact, two protection schemes are proposed in Section VI, and their performance is verified by simulation. This paper also focuses on the inference and bit-flips are introduced as the fault model in Section VII. Section VIII provides the error analysis and comparisons between FP and SC implementations. In Section IX, parity-based fault-tolerant schemes are proposed for both single-bit and multi-bit errors; they are evaluated and compared with methods found in the technical literature. Finally, the paper ends with conclusions in Section X.

II. Preliminaries

A. Triplet Networks

A typical Triplet Network (TN) consists of three identical weight-sharing subnetworks in training as shown in Figure 1 (a). Each subnetwork can be any feedforward network; this paper considers both MLP and CNN implementations. As introduced previously, TNs have been shown to have an excellent performance [5], [15] for classification when there is a paucity of training data available. Next, the training and inference processes of a TN are introduced.

1) Training process

During the training process (Figure 1 (b)), the three subnetworks of a TN accept the so-called triplets, including anchors, positive samples with the same class, and negative samples with 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

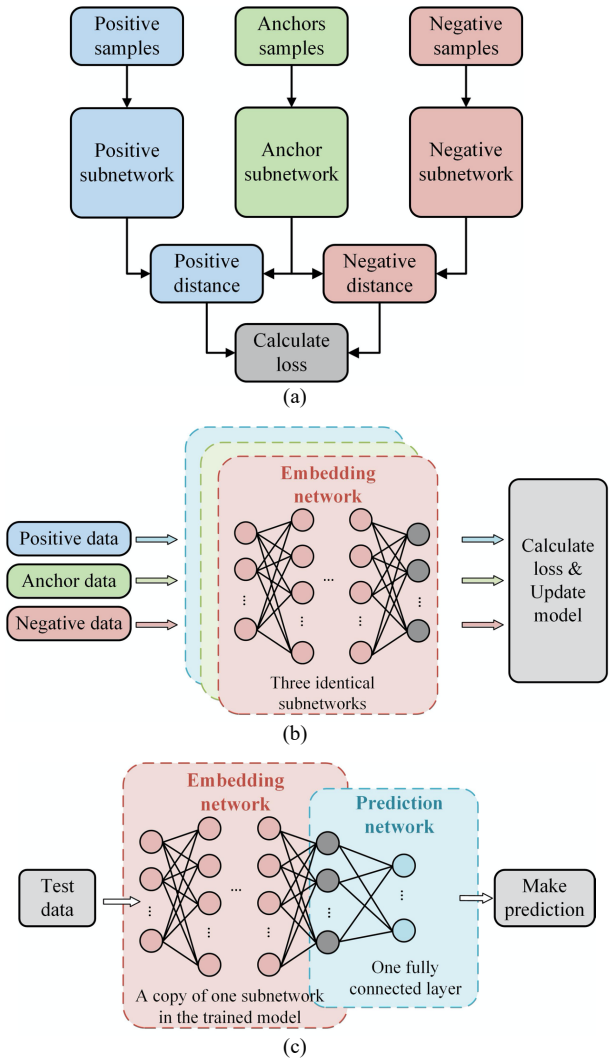


Fig.1. A triplet network: (a) structure; (b) training; (c) inference.

iteratively applying optimizers (such as gradient descent) to the loss for achieving 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

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

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 $d_+ - d_-$ [16]. 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 maps 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 simply realized by simpler schemes such as Support Vector Machine, logistic regression, or K Nearest Neighbors. This paper uses a one-hidden-layer MLP as the prediction network.

2) Inference process

The trained model of a TN can map the inputs into an embedding space in which samples from different classes can be distinguished; it is then used in the inference process to classify the test dataset.

One unknown sample (instead of triplets) is input to the network each time; so, only one of the three identical subnetworks in the trained model is employed. Let the output of the hidden units in the r-th layer be denoted as $y^r = [y_1^r \; y_2^r \; \cdots \; y_{m_r}^r]$. The output of the last layer is the feature embedding of the given input; the trained prediction network is applied for predictions. The inference process of TNs is illustrated in Figure 2 (b).

B. Stochastic computing

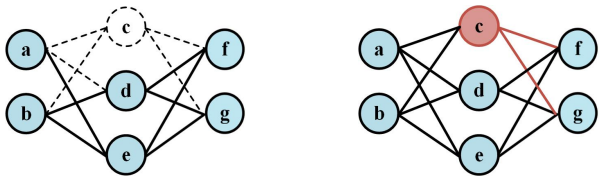
Stochastic computing (SC) is a promising solution to implement energy-efficient ANN accelerators [17], [18]. Different from conventional arithmetic, SC uses the probability of "1"s in a binary bitstream to represent a number. This feature permits that the SC design requires less hardware; for example, the multiplication in bipolar representation[1] can be simply realized by an XNOR gate.

This paper focuses on another advantage of SC, namely fault tolerance. Since each bit in the stochastic bitstream has the same significance, the SC arithmetic has a better resilience to a bit-flip error than conventional deterministic implementations [19]. In this paper, the SC implementation has been also evaluated as an effective fault tolerant approach for TN inference and compared with conventional TNs using the FP format.

III. Training Process: Stuck-at Fault Model

Since the network parameters are changing at each iteration of training, transient faults (such as random bit-flips) barely affect the result of the training process. Therefore, stuck-at faults are considered as the fault model during the training process. This model has been originally defined for permanent faults/defects of high/low logic values at physical transistor and circuit levels in Very Large Scale Integration (VLSI) [6]; for ANNs, the model has also been extended as an abstraction as per its impact [14]. As shown in Figure 2, the stuck-at faults in ANNs can be divided into two types: stuck-at 0 faults (missing the neuron) and stuck-at 1/-1 faults (saturated the neuron).

[1]This paper applies the SC arithmetic in bipolar representation, which encodes the values as $(2N_1 - N)/N$, where N is the length of the stochastic sequence and N_1 is the number of "1"s in it. For more explanation of SC, please refer to [17], [18].



(a) (b)

Fig.2. Examples of faulty network (with faults in neuron c). (a) Stuck-at 0 faults, missing the neuron; (b) Stuck-at 1 or -1 faults, saturating the neuron.

- ***Stuck-at 0 faults:*** A fault causing a missing neuron means that no signal (and data) is generated irrespective of the input. As it has been abstracted that the output is stuck at an intermediate value [13], the intermediate value after the activation function "*tanh*" (which is used in this paper) can be treated as "0", because the outputs are with a zero-mean normal distribution. For simplification, the fault causing a missing neuron is denoted as "stuck-at 0 faults" [11].
- ***Stuck-at 1/-1 faults:*** In a saturated neuron caused by faults the data or control line is held permanently high or low [7]. Therefore, it can be mapped to the scenario in which outputs of specific neurons are stuck at a maximum or a minimum value. Also, as activation functions (e.g., "*tanh*" with an output range of [-1, 1]) are applied after every neuron, this model can be simplified as stuck-at 1 or -1.

The execution of the training process in TNs with specific faulty neurons can be represented as patterns [13]. As the exhaustive testing for all possible fault-injection cases is prohibitive, then for simplification purposes, this paper assumes that all faults are injected into the same layer and the same subnetwork. Hence, the pattern representation is 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, $v \in \{0, 1, -1\}$.

IV. Impact of Faults on Training: Analysis

The over-provisioning of neurons in fully-connected neural networks provide an inherent fault-tolerant capability; a limited number of faults usually leads to a small accuracy loss or graceful degradation. However, stuck-at faults in the negative subnetwork of a TN can lead to an unexpected system failure as the network converges to a false solution (satisfying the constraints of the objective function as mentioned in Section II-A), so failing in the classification task. A theoretical analysis of such false solutions is presented in the following.

A. Stuck-at 0 faults in output layer

The incorrect convergence with stuck-at-0 faults in the negative subnetwork is rather intuitive when the faults are in the output layer of MLPs or the last fully connected layer in CNNs (i.e., $r = l$). Due to these faults, a spurious solution can be easily achieved in training (so it satisfies the constraints, but it is not a valid solution for the classification problem).

Let I be the set of indices of the stuck-at 0 faults and i denote the index of the faulty neurons, $i \in I$. The false solution only needs to map the anchor and positive subnetworks to similar

outputs, while the values of the neurons at the indices of the faulty units ($y_{a,i}^l$) are large enough (the absolute value can be close to 1 after $tanh$). Consider that the stuck-at 0 faults are injected, making $y_{n,i}^l = v_i$; as per (3), the constraint of the negative distance in this case is given by

$$d_- = \sqrt{\sum_{\substack{i=1,\\ i\notin I}}^{m_l} \left(y_{a,i}^l - y_{n,i}^l\right)^2 + \sum_{i\in I}\left(y_{a,i}^l - v_i\right)^2} \geq \sum_{i\in I}\left|y_{a,i}^l - v_i\right| > \varepsilon_- \quad (6)$$

and it can also be satisfied by even only considering the faulty terms $\left|y_{a,i}^l - v_i\right|$, $v_i = 0$.

The spurious solution is easy to be achieved; for the anchor and positive subnetworks, the weights address the common part of their inputs, and the negative subnetwork is ignored. So, the training converges at that solution in very few iterations.

B. Stuck-at 1/-1 faults in output layer

Let *I* be the set of indices of the stuck-at 1/-1 faults. The analysis of false solutions is similar to the case of stuck-at 0 faults. Consider (6) for $v_i = 1$ or -1; the constraint of d_- can also be satisfied by only considering the faulty terms $\left|y_{a,i}^l - v_i\right|$.

However, it could be worse because 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 a false solution from the start. For example, the L_{TL} in (4) can be represented as

$$L_{TL} = max(d_+ - d_- + M, 0)$$
$$= max\left(d_+ - \sqrt{\sum_{\substack{i=1,\\ i\notin I}}^{m_l} \left(y_{a,i}^l - y_{n,i}^l\right)^2 + \sum_{i\in I}\left(y_{a,i}^l - v_i\right)^2} + M, 0\right)$$
$$\leq max\left(d_+ - \sum_{i\in I}\left|y_{a,i}^l - v_i\right| + M, 0\right) \quad (7)$$

where $v_i = 1$ or -1. The faulty term leads to a false solution with zero-loss if $\sum_{i\in I}\left|y_{a,i}^l - v_i\right| \geq d_+ + M$. This can be achieved under a large number of faults. In practice, the initial outputs v_i are usually close to 0 due to weight normalization, the margin M is usually set to 1 and $|d_+| < 1$; so it is likely that the training gets stuck in the zero-loss situation at the beginning.

C. Stuck-at faults in hidden layers

The previous false solution (for stuck-at 0/1/-1 faults) 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

$$y_j^{r+1} = \phi\left(\sum_i w_{j,i}^r \cdot y_i^r + b_j^r\right), \quad (8)$$

where $w_{j,i}^r$ and b_j^r are the weight and bias; ϕ is the activation function ("*tanh*" used in this paper). The propagation between convolutional layers can be represented as

$$y_{u,v}^{r+1} = \phi\left(\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} y_{i+u,j+v}^r \cdot k_{i,j}^r + b_j^r\right) \quad (9)$$

where $k_{i,j}^r$ is the corresponding entry of the 180° rotated $n \times n$ convolutional kernel K^r (only enabled when $i \geq 0$ and $j \leq n$). The pooling layers do not affect fault propagation. 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 transmit to the following layers (so, no more concentrated on those faulty indices, but separate to all units). However, $tanh(x)$ bounds its inputs to the range $[-1, 1]$, and weakens the dominance of the large distances caused by faults. This behavior stacks over multiple layers; so when the faults occur in the early layers, the negative distance d_- at the spurious solution tends to be smaller. It is possible that $d_- < \varepsilon_-$ and this spurious solution becomes only a local minimum; in this case, the network can sometimes avoid it and continue to converge to the global solution (as the behavior in the fault-free cases). However, with an increasing number of faults, the network is more likely to be stuck at the false solution.

TABLE I
DETAILS OF DATASETS AND INFERENCE ACCURACY IN FAULT FREE CASE

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%

V. IMPACT OF FAULTS ON TRAINING: EVALUATION

This section studies the impact of faults on the training of a TN and its inherent fault tolerance using simulation. The purpose is to investigate the relationship between faults and the network's performance; hence, the classification error is used as the criterion. The stuck-at faults are applied as the fault models. Diverse scenarios, including the number and positions of faults, are considered. Four popular image classification datasets (MNIST [20], Fashion-MNIST [21], Cifar-10 [22], and SVHN [23]) are used; details of the datasets and their inference accuracy in the fault-free case are provided in Table I.

In the first subsection, the sensitivity of the TN is discussed for its anchor and positive subnetworks. The special behavior when the faults are in the negative subnetwork is discussed in the second subsection. The MLP implementation is used initially, and the results for the CNN implementation are provided in the last subsection.

A. Fault tolerance of the anchor/positive subnetwork

Next, the fault tolerance of TNs is evaluated when the stuck-at faults are in the anchor or positive subnetworks. The TN employs six-layer MLPs as subnetworks. The size and structure of the TN are chosen as the one that achieves the lowest classification error. The results of each simulation are trained for the same 20 epochs and averaged over 200 repeated trials.

Sensitivity on layer and number of faults: The dimensions of the layers (after the input layer) are represented by the vector $[m_1\ m_2\ m_3\ m_4\ m_5] = [1024\ 512\ 256\ 128\ 128]$. The percentage of faulty neurons t is varied from 0 to 100%, the number of faults in the r-th layer is $f = m_r \times t$ (rounded). Figure 3 shows the simulation results with the pattern $Q_T = (f = m_r \times t, r \in \{1, \cdots, 5\}, c \in \{anchor\}, v \in \{0\})$, $t \in [0, 1]$.

From these results, the optimal network configuration (achieving the lowest classification error in the error-free cases) shows a very strong resilience, so a no accuracy loss with a small number of faults. The performance of the network degrades with an increasing number of faults, but only with a very large percentage of faulty neurons. The latter layers

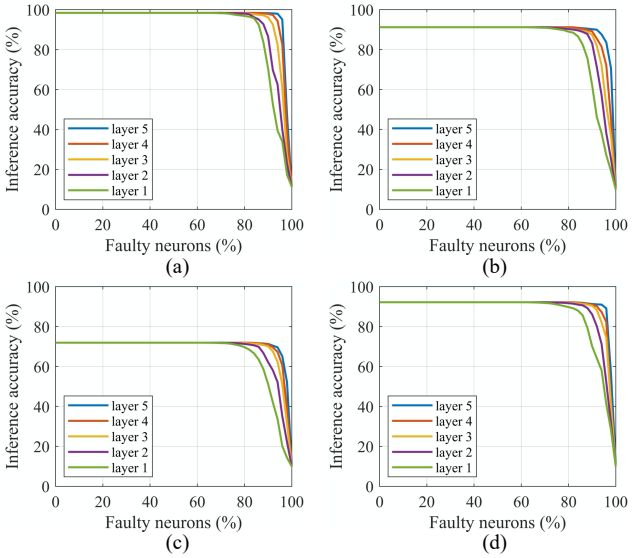


Fig.3. Inference accuracy of different percentage of faulty neurons in each layer with dataset (a) "MNIST"; (b) "Fashion-MNIST"; (c) "CIFAR-10"; (d) "SVHN".

generally have better fault tolerance compared with early layers; for example, when considering the plots for the output layer (layer 5), the accuracy is very close (degradation less than 0.1%) to the fault-free case for each dataset even with 90% faulty neurons. So, the TN still operates reliably with most neurons being stuck-at faults in a single layer. Therefore, as so many faults are not likely to happen in practice, the stuck-at faults in anchor/positive subnetworks of a TN only lead to trivial effects that are compensated by the training process.

Sensitivity on subnetwork and type of faults: 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 and the percentage of faulty neurons is $t = 90\%$. Figure 4 shows the simulation results with the pattern $Q_T = (f = m_r \times \mathrm{t}, \mathrm{r} \in \{5\}, \mathrm{c} \in \{\mathrm{anchor}, \mathrm{positive}\}, v \in \{0,1,-1\}), t = 0.9$. In this figure, "ST0" denotes the missing hidden units; "ST1" and "ST-1" denote the hidden units that are stuck at the maximum or minimum, respectively. The dashed lines in the figures are the accuracy in the fault-free case.

The results show that the anchor subnetwork is more sensitive to faults than the positive subnetwork. 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. Therefore, it is expected to have slightly better fault tolerance in the positive subnetwork.

The TN has a better resilience to "ST0" faults than "ST1" and "ST-1" faults. This means 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 0 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).

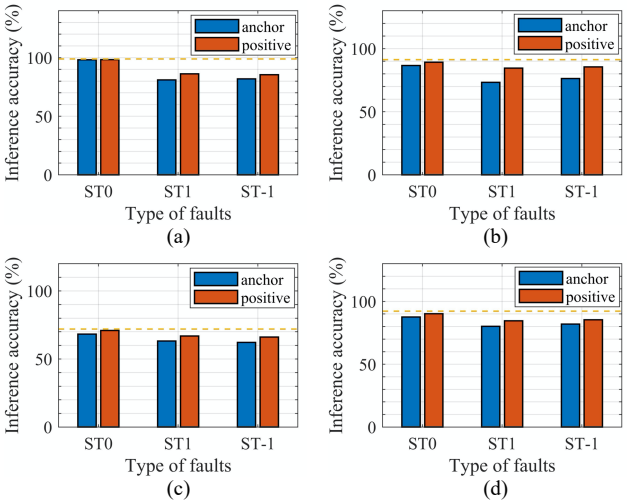


Fig. 4. Inference accuracy of different types of faults into the anchor and positive subnetworks with dataset (a) "MNIST"; (b) "Fashion-MNIST"; (c) "CIFAR-10"; (d) "SVHN".

B. Fault tolerance of the negative subnetwork

The previous analysis suggests that faults in the negative subnetwork can lead to fatal system failure due to incorrect convergence. This is evaluated in the following; note that this behavior may be related to the number of faults and the faulty layers. Therefore, the different percentages of faulty neurons in each layer (with dimensions $[m_1\ m_2\ m_3\ m_4\ m_5] = [1024\ 512\ 256\ 128\ 128]$) of the negative subnetwork are assessed during the training process. It is assumed that the fault type is stuck-at 0 faults (stuck-at 1/-1 faults reach a zero-loss from the beginning according to Section IV-B). Figure 5 shows the simulation results with the pattern $Q_T = (f = m_r \times t, r \in \{1, \cdots, 5\}, c \in \{negative\}, v \in \{0\}), t \in [0, 0.05]$.

As per Figure 5, a significant accuracy loss can be caused by only a few faults in the negative subnetwork. An accuracy close to 10% implies that the model completely fails in classification, so it converges to a false solution. This is different from the strong resilience of the anchor and positive subnetworks discussed in Section IV-A. Such behavior is in line with the analysis that the network converges to a spurious solution; the accuracy tends to be higher when the erroneous units are in the early layers. Especially for dataset "MNIST", when the faults are in the first layer of the negative subnetwork, performance is very close to the fault-free case (if the training is prolonged, some other cases can also achieve high accuracy). It can be confirmed by the analysis that the network can overcome the spurious solution and continue the correct convergence; this happens when such a false solution becomes not strong enough after being weakened by the activation functions.

Moreover, the inference accuracy decreases as more faults are injected in the negative subnetwork. The network has a lower probability to escape from the spurious solution (local minimum), so agreeing with the analysis. Especially, when the number of faults is large enough (such as $f > 5\% \times m_r$), the network tends to be completely stuck at the false solution even when the faults are in the early layers.

C. Fault tolerance with CNN implementation

The previous simulation results have employed a TN with MLPs as subnetworks; this section extends it to CNN

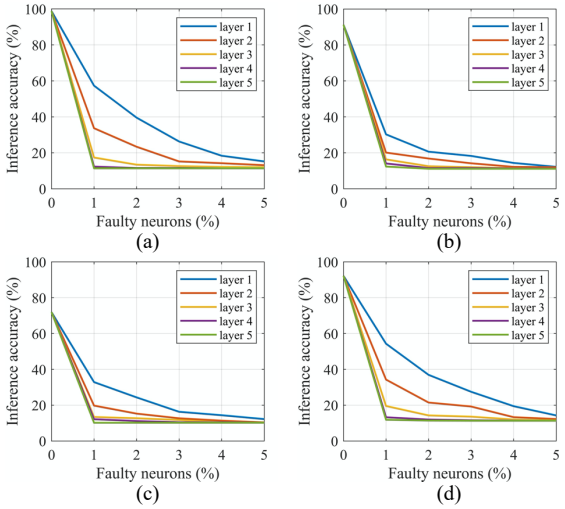


Fig. 5. Inference accuracy of different percentages of faulty neurons in the negative subnetwork with dataset (a) "MNIST"; (b) "Fashion-MNIST"; (c) "CIFAR-10"; (d) "SVHN".

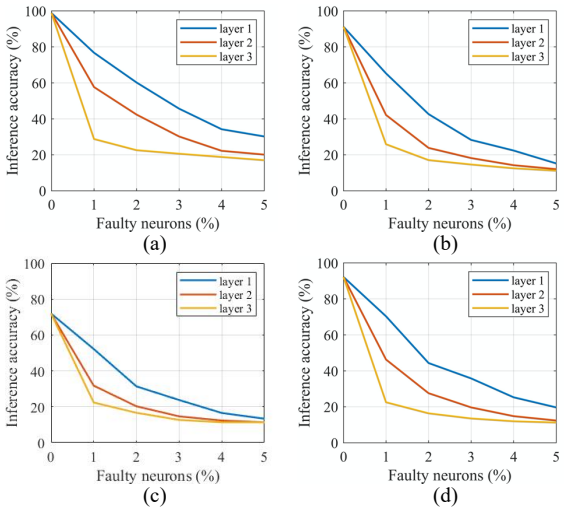


Fig. 6. Inference accuracy of different percentage of faulty neurons in the negative subnetwork (CNN version) with dataset (a) "MNIST"; (b) "Fashion-MNIST"; (c) "CIFAR-10"; (d) "SVHN".

implementation. Each of these subnetworks consists of three convolution layers (with output channels and sizes of 16×24×24 - 32×16×16 - 64×8×8) and two fully-connected layers; *tanh* is used as the activation function and max pooling is applied. The datasets and other configurations in the TN are the same as in the MLP-based version. This simulation tests the stuck-at faults in the convolutional layers; since CNNs usually share weights and computational units between channels, faults are injected equally into all channels in the same layer. The dimensions of the convolutional layers (each channel) are represented by the vector $[m_1 \ m_2 \ m_3] = [576 \ 256 \ 64]$.

From the results for all datasets, it is confirmed that the CNN-based version also has strong resilience to the stuck-at faults in the anchor or positive subnetworks. As per the simulation results, the CNN-based TN is still reliable (degradation less than 0.1%) with up to 89.66% of the neurons in a single layer with stuck-at faults. Also, the special case when the faults are in the negative subnetwork is observed in the CNN implementation. Figure 6 shows the simulation results in the three convolutional layers with the pattern $Q_T = (f = m_r \times t, r \in \{1,2,3\}, c \in \{negative\}, v \in \{0\}), t \in [0, 0.05]$.

The conclusions are like for the MLP implementation discussed in Section IV. Compared with the results in Figure 5, the CNN version performs slightly better; however, the faults in the negative subnetworks still lead to a significant accuracy loss and it tends to be worse when the erroneous units are in the latter layers.

VI. Proposed Fault Tolerant Methods for Training

The previous analysis and simulation results suggest that fault tolerant schemes are required for dealing with stuck-at faults in the negative subnetwork of TNs. Next, we propose two methods to protect the training process from missing or saturating neurons (i.e., against stuck-at 0 and stuck-at 1/-1 faults) respectively. Also, the proposed methods are evaluated by simulations to prove their efficacy.

A. Regularization to anchor outputs

From the analysis in Section IV-A, the false solution with stuck-at 0 faults occurs when the faulty neurons in the anchor and positive subnetworks have large absolute values. In this case, the constraint of the negative subnetwork can be simply satisfied by the faulty term.

To prevent an incorrect convergence towards large outputs in the anchor or positive terms, an extra penalty term (i.e., the L2-regularization term on the anchor outputs from different layers) must be added to the loss function as

$$L = max(d_+ - d_- + M, 0) + \lambda \sum_{j=1}^{l} \sum_{i=1}^{m_j} {y_{a,i}^j}^2 \qquad (10)$$

where m_j is the dimension of layer j and λ is the parameter determining the influence of the penalty term. This paper assumes that faults occur in any layer; however, if the faulty layer can be specified, only regularization of the corresponding outputs is required. Consider TL as an example only (applicable to other functions too). The regularization term can be added to either the anchor or the positive subnetworks (their outputs tend to be the same due to the objective function). In this paper, the regularized anchor outputs affect both positive and negative distances, so they can better accelerate convergence.

The regularization term on the anchor outputs can restrict the faulty units from taking a large value, therefore preventing the convergence to the false solution. By choosing a proper parameter λ, the additional term is negative to the error-free case; however, it prevents an unexpected system failure to occur when caused by the stuck-at 0 faults in TNs.

B. Modified margin

As per the analysis in Section IV-B, measures should be taken to prevent training to be in a zero-loss situation from the start under stuck-at 1/-1 faults. In this case, let's consider the objective function TL in (7) as the example; the difference caused by the faulty terms $\sum_{i \in I} |y_{a,i}^l - v_i|$ must be compensated. An intuitive way is to increase the original margin M; let the number of faulty neurons be denoted by f, so increasing the margin by f can effectively guarantee that the training starts correctly (as the initial outputs $y_{a,i}^l$ are usually close to 0).

By the analysis in Section III, convergence can still be in the false solution of the subsequent training process; in this case, the regularization of the loss function is no longer feasible. A possible reason is that the regularized outputs towards 1 or -1

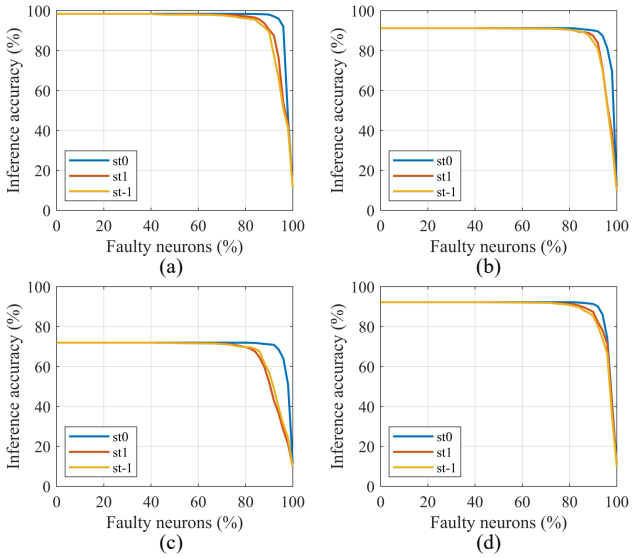


Fig. 7. Inference accuracy of different percentage of faulty neurons with stuck-at 0 faults (regularization on anchor outputs), and stuck-at 1/-1 faults (modified margin): (a) "MNIST"; (b) "Fashion-MNIST"; (c) "CIFAR-10"; (d) "SVHN".

can cause vanishing gradients with bounded activation functions. Therefore, the method of further increasing the margin is considered; for each faulty term $\left|y_{a,i}^{l} - v_i\right|$, the maximum error is 2 when $v_i = 1$ or $v_i = -1$, $-1 \le y_{a,i}^{l} \le 1$. By (7), 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 entire training process.

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 [24]; setting the margin to an arbitrarily large value slows down the convergence in the error-free case. Therefore, increasing the margin must be thought carefully. We propose to modify the margin at the beginning of the training by:

$$M' = M + 2\lfloor d\_ \rfloor \tag{11}$$

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 error-free cases (valid for all applied datasets); thus, it can be an estimate of the number of (stuck-at 1 or -1) 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 false solutions when stuck-at 1/-1 faults are present in the negative subnetwork.

C. Evaluation

This subsection provides the simulation results for the proposed technique of regularization to anchor outputs (modified margin) to avoid false solutions when the stuck-at 0 (stuck-at 1/-1) faults are in the negative subnetwork. The fault injection process and the network configurations are the same as those applied in Section V. Simulation has been performed for faults that are randomly injected into different layers, so also shows the worst case (when faults are in the output layer). Consider the MLP implementation as an example with the network configuration of Section V-A. Also, the pattern of the fault injection simulation is given by $Q_T = (f = m_r \times t\,, r \in \{5\}, c \in \{negative\}, v \in \{0, 1, -1\}), t \in [0, 1]$.

The inference accuracy at different percentages of faulty neurons is shown in Figure 7. Compared with the previous results in Section V, the negative subnetwork is no longer sensitive to stuck-at faults. The curves are now like those for the anchor/positive subnetworks, so the false solutions can be effectively avoided by the proposed approach. More generally, the regularization on the anchor outputs for stuck-at 0 faults operates very well; increasing large margins for stuck-at 1 or -1 faults leads to some accuracy loss, but performance is still satisfactory. Therefore, the proposed fault-tolerant methods can be utilized in TNs to avoid fatal system failures caused by stuck-at faults. Also, they can be applied simultaneously to protect the network from all other types of stuck-at faults.

VII. Inference Process: Random Bit-flip Model

During the inference process, the fault model is random bit-flips; these faults are transient in nature, and they have been widely accepted as random upset values in memory. Therefore, in the fault model, random bit-flips can occur at any of the bits by reversing the value (0 to 1, or 1 to 0); since predictions are directly determined by the weights stored in memory, bit-flip faults are critical in the inference process.

When more than one random bit-flip occur (mostly caused by a radiation particle also known as soft errors), the affected memory cells are usually physically close; so, a soft error tends to either affect multiple data bits in a single memory word, or single data bits in multiple adjacent words [25]. Therefore, two scenarios caused by bit-flip faults are considered: 1) single-bit error: multiple errors occur but each faulty weight/memory word has only one bit flipped; 2) multi-bit error: multiple errors occur but they affect the same weight/memory word.

Next, an analysis is provided for the weights in FP format as affected by bit-flips. The implementation with SC is known to have inherent resilience to errors; hence, it is also analyzed and compared to provide insight into the fault tolerance of FP TNs.

A. Fault model for FP format

This paper assumes that the weights of the TNs are in the IEEE 754 standard single-precision floating-point format [26], i.e., 1 sign bit, 8 exponent bits, and 23 mantissa bits. For single-bit errors, any one of those bits is flipped to the reversed state.

Similarly, the execution of the inference process in TNs with bit-flip faults can be represented as patterns. The pattern is represented by $Q_I = (f, r, P)$, where f is the number of faults and r denotes the layer to be injected, $f < m_r$; the set $P = \{p_1, p_2, \cdots, p_f\}$ denotes the position of the corresponding faults, so the location of the bit-flip at the p_i-th bit, $p_i \in \{1, 2, \cdots, 32\}$. The decimal value of a floating number is given by

$$x = (-1)^S \times 2^{E-127} \times (1 + M \times 2^{-23}), \tag{12}$$

where S, E, and M are the decimal number represented by the sign, exponent, and mantissa bits; so, faults on the sign or exponent bits ($p_i \le 9$) lead to more significant changes.

Let x_e denote the erroneous value represented by a single flipped bit of an FP number x. If the fault occurs in the sign, exponent, or mantissa bits, the erroneous value and the absolute error are represented as

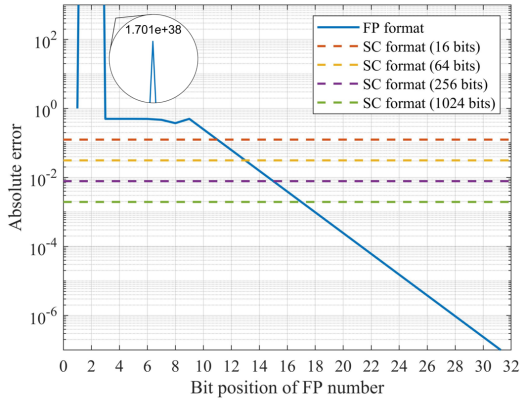


Fig. 8. Comparison of absolute error in FP (32-bits) and SC formats with one bit-flip fault for example weight value 0.5.

$$x_e = \begin{cases} -x, & \text{sign bit;} \\ 2^{E_i}x, & \text{exponent bits;} \\ \frac{M_i M + 2^{23}}{M+2^{23}}x, & \text{mantissa bits;} \end{cases} \quad (13)$$

$$|x - x_e| = \begin{cases} |2x|, & \text{sign bit;} \\ |(2^{E_i}-1)x|, & \text{exponent bits;} \\ \left|\frac{(M_i-1)M}{M+2^{23}}x\right|, & \text{mantissa bits,} \end{cases} \quad (14)$$

where $E_i = 2^i$ if the i-th bit (counted from least significant bit (LSB) to most significant bit (MSB)) of the exponent flipped from 0 to 1; $E_i = 2^{-i}$ if the i-th bit of the exponent flipped from 1 to 0. Similarly, $M_i = 2^i$ when the i-th bit of the mantissa is flipped from 0 to 1; otherwise $M_i = 2^{-i}$. M is the decimal number represented by the mantissa bits in (12); it is concluded by (14) such that the error of an FP number is related to its absolute value $|x|$. Therefore, bit-flips in a larger FP number lead to a more significant error.

B. Fault model for SC format

An SC number has the feature that each bit has the same significance, this is an advantage from the perspective of fault tolerance. Considering the same pattern of bit-flip fault to the inference process as in the FP TNs, the absolute error an N-bit stochastic sequence with bipolar representation is given by

$$|x - x_e| = 2N^{-1}. \quad (15)$$

The error of an SC number caused by a bit flip relates to only the length of the stochastic sequences; hence, it can be treated as a constant because the length N is a predefined parameter.

C. Comparison

To compare the tolerance of FP and SC formats to the bit-flip fault in the weights, the corresponding absolute value of the erroneous element is required. Even though the weight distribution can vary greatly from different applications, they usually concentrate within a limited range. For TNs with normalized datasets (as used in this paper), the weight distributions are approximately normal distributions with zero mean and 0.6 variance. To give an intuitive illustration, consider an average absolute weight value of 0.5 as an example (represented in FP as: sign bit "0", exponent bits "01111110", and all zeros as mantissa bits). Figure 8 illustrates the absolute error caused by one flipped bit on different positions (the x-axis is arranged in the order of sign bit: 1; exponent bits: 2-9; mantissa bits: 10-32) with FP and SC formats. Different lengths of SC sequences are compared and denoted as horizontal lines in this figure.

Figure 8 shows that the errors on the mantissa bits of FP numbers are relatively small and decrease exponentially. The error when the fault occurs on any position of an SC number is also very small. The intersections show that the error at the 11/13/15/17-th bit is equivalent to an error in SC format with 16/64/256/1024 bits. One noticeable finding is that a 0 to 1 flip in significant exponent bits can lead to extremely large errors (such as the 2nd bit in Figure 8). Also considering (14) and (15), errors in FP format increase with a larger weight, while the errors in SC format remain unchanged.

The above analysis quantifies the errors in FP and SC formats; the FP format suffers significant errors when the faults occur in the MSBs (especially a 0 to 1 flip), while the SC format keeps a constant small error with faults in any position. However, it does not directly show the implications by which the inference accuracy of the TN is affected by these errors. The next section provides further evaluations of the impact of faults.

VIII. Impact of Faults on Inference: Evaluation

This section deals with simulation as fault injection during the inference process of TNs implemented in both FP and SC format. The random bit-flip is applied as the fault model; the network configurations and the datasets are the same as in the previous section. The results of each evaluation are averaged over 10000 repeated trials.

A. TNs with FP format

For the TNs with FP format, different numbers of faults in the TN have been considered during the inference process. Both cases of single-bit and multi-bit errors have been assessed.

Single-bit error: A faulty bit is injected into each weight and multiple erroneous weights can be present. Different from training, a fixed number of faults is assumed and they are randomly injected into all layers. The patterns Q_l of this fault injection process is represented as $Q_l = (f \in \{1,2,\cdots,10\}, r \in \{1,\cdots,5\}, P)$. With r and $P = \{p_1, p_2, \cdots, p_f\}$ uniformly and randomly chosen within the feasible range, faults occur in any layer or bit position; the fault-free case $f = 0$ is included for comparison. The inference accuracy is plotted in Figure 9 (a).

The results show the accuracy constantly decreases with the number of faulty bits for all datasets, so the inference accuracy and the number of faults have a linear relationship. Different from the fault injections for the training process, only a few weights with single-bit errors lead to an obvious degradation. The inference process of TNs is sensitive to random bit flips; also, by monitoring the position of the faults in all trials, 97.2% of the degradation (defined as an accuracy loss larger than 0.01%) occurs when the bit-flip (on at least an erroneous weight) is on the exponential bits.

Multi-bit error: This experiment injects a pattern consisting of multiple faulty bits into only one weight; this pattern is nearly identical to the previous experiment. The only difference is that all bit-flips are injected to the same erroneous weight. The inference accuracy is shown in Figure 9 (b).

Similar to the previous case, the accuracy loss is linearly proportional to the number of faulty bits and only one erroneous weight with multiple bit-flips can lead to significant degradation. Thus, the same importance should be attached to

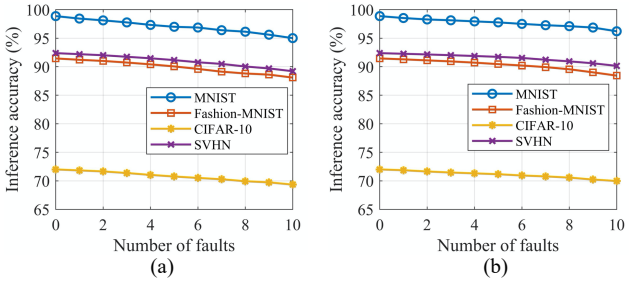


Fig.9. Inference accuracy of different number of faults injected in the inference process (FP implementation) for (a) single-bit error; (b) multi-bit error.

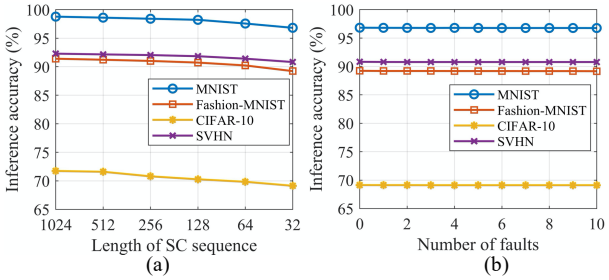


Fig.10. Simulation results for TNs with SC format. The inference accuracy of (a) different SC sequence lengths; (b) different number of faults.

the case of multi-bit error; also, 95.8% of the degradation in all the repeated trials involves at least one bit-flip on the exponential bits of the erroneous weight.

B. TNs with SC format

This section provides the simulation results of fault injection in the TN with SC arithmetic during the inference process. The design of the SC-based inference network has been described in [18]. The goal is to support the analysis for TNs with FP format because the SC implementations are generally considered tolerant to bit flips. This simulation can also reveal the relationship between errors caused by faults and the predicted accuracy loss. For the worse cases, multi-bit errors are applied in this subsection.

Sensitivity on sequence length: Different lengths of stochastic sequences in SC TNs are utilized in inference. Since the bit length is different from the FP format, the fault rate is set equivalent to the one used in Section VIII-A (a multi-bit error with 10 faulty bits). The pattern of this fault injection can be thus represented as $Q_I = (f = \frac{10N}{32}, r \in \{1, \cdots, 5\}, P)$, f is rounded off. Sequence lengths from $N = 1024$ bits to $N = 32$ bits are utilized; the results are plotted in Figure 10 (a).

Sensitivity on the number of faults: The first simulated scenario relies on different numbers of faults, but it assumes that all faulty bits are of the same weight. The bit length of the stochastic sequences is set to $N = 32$; the pattern of this fault injection is identical to Section VIII-A as $Q_I = (f \in \left\{\frac{N}{32}, \frac{2N}{32}, \cdots, \frac{10N}{32}\right\}, r \in \{1, \cdots, 5\}, P)$. The results are given in Figure 10 (b).

Figure 10 (a) shows an accuracy loss with shorter SC sequences; however, the accuracy remains almost unchanged in Figure 10 (b) when increasing the number of faulty bits; this phenomenon does not relate to different impacts for errors (illustrated in Figure 8), but it is caused by the less accurate SC arithmetic with shorter sequence lengths. In Figure 8, the SC format does not show an immediate improvement over the FP format in terms of absolute error, but its inference accuracy shows a significant advantage. As a fault-tolerant implementation, SC has the feature that all bits share the same significance, so bit-flip faults cannot lead to significant changes. The impact of errors with the SC format ($N = 32$) is similar to the mantissa bits of the FP format; while Figure 9 (b) and Figure 10 (b) show different plots at the same fault rate, the degradation is mostly caused by the changes in the exponent bits. If the large outliers can be eliminated, the fault tolerance of TNs with the FP format can be greatly increased (compared to the performance of SC implementations).

IX. Proposed Fault Tolerant Methods for Inference

The previous analysis and simulation by error injection have shown the effects of random bit-flip faults on the inference process of the TNs; in particular, large outliers in weights are caused by flips in the significant bits. This paper focuses on low-cost solutions while still retaining satisfactory performance and taking advantage of the inherent fault tolerance of the NNs. The proposed methods protect the inference process from bit-flips by inserting parity bits for both single-bit and multi-bit errors. Also, related works are discussed, and the proposed methods are evaluated to show their efficiency.

A. Replacing LSB by a parity bit for single-bit errors

As per the error injection process, the protection of MSBs (exponent bits) is of great importance and can greatly reduce the loss of accuracy. A single parity check is widely used for error detection and thus, it is applied in this paper. A parity bit is employed to check the exponent bits (encoded by the XOR operation with each bit) as illustrated in Figure 11; therefore, the parity bit detects a single-bit error when any of the significant bits are changed by faults. Once the bit-flip is detected, the corresponding weight is directly set to zero to prevent outliers; moreover, the parity bit replaces the LSB of the mantissa. This scheme does not require additional memory overhead and shows the applicability of the proposed scheme because the data format remains unchanged. Setting the weight to zero and replacing the mantissa bit can lead to a loss in accuracy; this is usually negligible compared to the benefits of protecting the MSBs. With normalization and activation functions (such as *sigmoid* or *tanh)*, the weights in NN applications usually have small absolute values (nearly close to zero); the effect of a changed mantissa bit is verified by the analysis in Section VII-A and related literature [27]. For example, in Figure 8, the error due to the parity bit (32-th) is at a level of 10^{-7}; with large outliers avoided, the degradation caused by bit-flip faults is expected to be greatly mitigated and evaluation of performance is provided by simulation.

B. Replacing LSBs by parity bits for multi-bit errors

The case of multiple faults in one weight is rarer than the single case; however, as code-based protection for single-bit errors can fail, this paper also proposes a parity-based protection method for multi-bit errors.

The principle in the proposed methods is similar to detecting single-bit errors by using parity bits (inserted at the LSBs of the mantissa) to detect errors in the exponent bits. However, since more than one fault can occur in these bits, more parity bits are

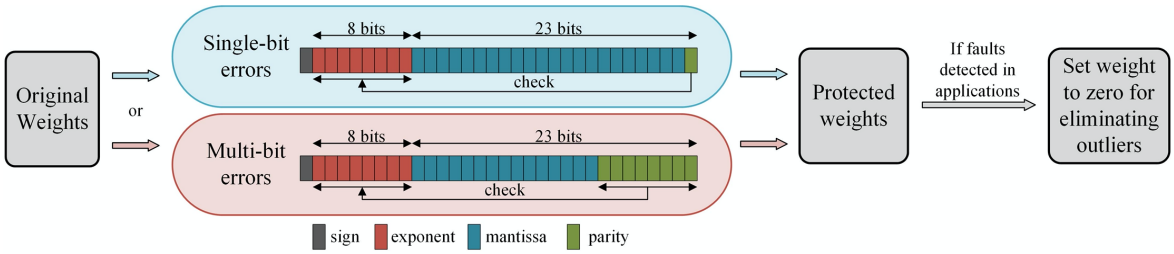


Fig. 11. Parity-based fault tolerance for bit-flips during inference process for single-bit or multi-bit errors.

required to check each of them (one parity bit encodes an exponent bit and 8 bits are required for an exponent in the single-precision FP format) as shown in Figure 11. Similarly, if a fault is detected, the weight is directly set to zero to prevent large outliners. Replacing more LSBs in the mantissa decreases the accuracy, but as per the analysis in Section VII-A, such loss is still negligible (<0.001% in fault-free cases) compared with a possible fatal effect due to these faults; for example, in Figure 8, the error caused by the 8 parity bits is at a level of 10^{-5}. Further evaluation is provided by simulations to verify the feasibility of the proposed methods with also a loss in accuracy.

C. *Related works*

Fault tolerance of NNs is required in critical applications and different approaches for protecting the model from bit-flip faults in weights during inference have been studied; most of these methods require some redundancy either spatial or temporal [11] or retraining the model by artificial fault injection [28]. Traditional single-error correction (SEC) codes can fully recover the faulty weights by utilizing 6 additional bits (for the 32-bit FP format). However, this overhead may not always be acceptable, especially for NN applications in hardware/power-constrained platforms, such as portable devices/systems.

Therefore, this paper focuses more on solutions for random bit-flips with low memory/hardware overheads. The inherent resilience of NNs to faults allows the use of low-cost methods with satisfactory classification error; for example, [29] proposed an optimization technique based on the observation that resilience is not homogeneous over the networks; so, the weights are adjusted according to a resilience prediction criterion. Similar to this paper, the replacement of weights with coding bits has been already proposed in the technical literature. In [30] an SEC code has been used to replace the 4 LSB in the mantissa to protect the sign and exponent bits; in [31] a similar idea to using parity bits has been proposed, but all memory bits for the weights are protected. However, those code-based methods are better suited for single-bit errors, and hence, solutions for multi-bit errors have to also be investigated; for example, [30] has applied filters to eliminate the large values in the weights caused by faults. So different methods are compared with the proposed parity-based fault tolerance for both single-bit and multi-bit errors.

D. *Evaluation*

The proposed methods and schemes of related works are compared with bit-flip faults in the inference of TNs. The MLP is used for subnetworks and the configurations are identical to those in Section VIII; the results are evaluated by the accuracy loss compared with the fault-free cases. The results with no fault tolerance are also provided for comparison.

TABLE IV
ACCURACY LOSS (%) OF FAULT TOLERANCE METHODS WITH DIFFERENT DATASETS FOR 10 SINGLE-BIT ERRORS

Scheme	MNIST	Fashion-MNIST	CIFAR-10	SVHN
Unprotected	3.765	3.157	2.460	2.967
[29]	0.662	0.582	0.449	0.560
Code in [30]	0.178	0.146	0.108	0.117
[31]	**0.007**	**0.005**	**0.006**	**0.006**
Proposed method	0.118	0.104	0.089	0.086

TABLE V
ACCURACY LOSS (%) OF FAULT TOLERANCE METHODS WITH DIFFERENT DATASETS FOR AN MULTI-BIT ERROR WITH 10 FAULTY BITS

Scheme	MNIST	Fashion-MNIST	CIFAR-10	SVHN
Unprotected	2.459	2.611	1.932	2.030
[29]	0.388	0.301	0.274	0.319
Filter in [30]	0.068	0.080	0.062	0.074
Proposed method	**0.050**	**0.056**	**0.043**	**0.053**

For the single-bit error evaluation, one bit-flip has been simulated for 10 different random weights; the results are averaged over 10000 repeated trials and are shown in Table IV. The method in [30] has the lowest accuracy loss, because it performs full correction by SEC; the proposed method eliminates on average 96.76% of accuracy loss caused by faults, so better performance than [29] and [31]. Considering that the proposed parity-based method is much simpler and replaces fewer bits, it can be thought as an effective alternative for TNs.

For multi-bit errors, 10 bit-flips are applied to one different random weight; the results are averaged over 10000 repeated trials as shown in Table V. The proposed method eliminates on average 97.74% of accuracy loss and again it performs better than [29] and the filter in [30]. The replaced bits in the mantissa are thus shown to have a marginal effect compared with the benefits of fault tolerance. Therefore, if the application is at risk from multi-bit errors, the proposed method provides comprehensive protection.

X. CONCLUSION

This paper has comprehensively studied the fault tolerance of Triplet Networks (TNs) under stuck-at faults in training and random bit-flips in inference.

For the training process, the analysis has revealed a special case: the stuck-at faults in the negative subnetworks can cause incorrect convergence when the loss is dominated by the faulty terms; moreover, stuck-at 1/-1 faults can even lead to a zero-loss situation at the beginning of training. These faults can make the network converge to false solutions, so resulting in system failure. Simulation results have been provided for evaluating the impact of stuck-at faults, by considering different number and position of faults and affected subnetworks. It has been

shown that the anchor and positive subnetworks are not sensitive to stuck-at faults. However, several faults in the negative subnetworks can lead to invalid training, which has confirmed the proposed theory of false solutions.

Two fault-tolerant methods have been proposed to solve the special case in the negative subnetworks. As per the outlined analysis, the loss function with regularization on the anchor outputs can be applied for stuck-at 0 faults, while the so-called modified margin scheme is a viable solution for stuck-at 1/-1 faults. Simulation has proved their performance and efficacy as false solutions can be effectively avoided; therefore, the proposed methods can be used for TNs during training to avoid a fatal system failure caused by stuck-at faults.

For the inference process, the theoretical analysis provided in this paper has shown that the flips in the exponent bits of the floating-point (FP) format lead to very large classification errors, while the impact of errors in the stochastic computing (SC) format is rather small, independently of the locations of the faulty bits. Comparison between the FP and SC formats in the analysis has been verified by extensive simulation; it can be concluded that the extremely large outliers in erroneous FP numbers are the main reason for degradation in accuracy. As with no such problem, SC implementations have shown strong resilience to random bit-flips; therefore, the fault tolerance of FP-based TNs can be improved by eliminating these outliers.

Two methods have also been proposed for protecting the TN from bit-flip faults during inference against both single-bit and multi-bit errors; they are based on the use of parity bits to check the exponent bits and prevent possible large outliers. These approaches lead to a small accuracy loss and they can efficiently improve the performance of faulty TNs without introducing memory redundancy. Evaluation has been pursued for the proposed methods and other approaches in related works. For single-bit errors, the proposed scheme performs better than all others except for SEC-based full correction (which requires a significantly larger memory overhead); for multi-bit errors, the proposed scheme achieves the lowest accuracy loss.

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." in *Proc. 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," in *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," *IEEE Trans. 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," In *Proc. 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," in *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," in *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," *IEEE Trans. on Emerging Topics in Computing*, pp. 1-15, Sept. 2021 (early acces).

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

[10] M. D. Emmerson and R. I. Damper, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE Trans. on Neural Network*, vol. 4, no. 5, pp. 788-793, Sep. 1993.

[11] T. Haruhiko, M. Masahiko, K. Hidehiko and H. Terumine, "Enhancing both generalization and fault tolerance of multilayer neural networks," in *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," in *Proc. 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 Network*, vol. 10, no. 3, pp. 539-553, 1997.

[14] C. H. Sequin and R. D. Clay, "Fault tolerance in artificial neural networks", in *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] Y. Liu and C. Huang, "Scene Classification via Triplet Networks," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 1, pp. 220-237, Jan. 2018.

[17] W. J. Gross and V. C. Gaudet, Stochastic computing: Techniques and Applications, Springer Internation Publishing, Feb. 2019.

[18] S. Liu, X. Tang, F. Niknia, P. Reviriego, W. Liu, A. Louri, and F. Lombardi, "Stochastic divders for low latency neural netoworks," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4102-4115, 2021.

[19] W. Qian, X. Li, M. D. Riedel, K. Bazargan and D. J. Lilja, "An Architecture for Fault-Tolerant Computation with Stochastic Logic," *IEEE Trans. on Computers*, vol. 60, no. 1, pp. 93-105, Jan. 2011.

[20] 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.

[21] 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.

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

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

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

[25] M. Maniatakos, M.L. Michael, Y. Makris, "Vulnerability-based interleaving for multi-bit upset (MBU) protection in modern microprocessors," in *IEEE International Test Conference*, pp.1-8, Nov. 2012.

[26] W. Kahan, "IEEE standard 754 for binary floating-point arithmetic", Lecture Notes on the Status of IEEE 754, no. 94720-1776, pp. 11, 1996.

[27] G. Li, S. K. S. Hari, M. Sullivan, et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications", *in Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1-12, Nov. 2017.

[28] C.-T. Chiu, K. Mehrotra, C. K. Mohan and S. Ranka, "Robustness of feedforward neural networks", in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 2, pp. 783-788, Apr. 1993.

[29] C. Schorn, A. Guntoro and G. Ascheid, "An Efficient Bit-Flip Resilience Optimization Method for Deep Neural Networks," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1507-1512, May 2019.

[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," *IEEE Trans. on Computers*, Jun. 2022.

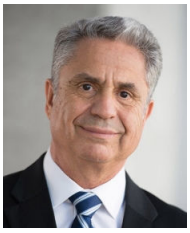
[31] M. Qin, C. Sun, & D. Vucinic, "Improving robustness of neural networks against bit flipping errors during inference," *Journal of Image and Graphics*, vol. 6, no. 2, pp. 1-6, Dec. 2018.

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 studying for the PhD degree in the Department of Electrical and Computer Engineering, Northeastern University. His current research directions are neural networks 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 the 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) received the Ph.D. degree in Microelectronics and Solid-State Electronics from Harbin Institute of Technology, Harbin, China, in 2018. She was a post-doctoral researcher with the Department of Electrical and Computer Engineering (ECE), Northeastern University, Boston, USA, from 2018 to 2021, and is currently an Assistant Professor with the Klipsch School of ECE, New Mexico State University, Las Cruces, USA. She serves as an Associate Editor for the IEEE Trans. on Emerging Topics in Computing and the IEEE Trans. on Nanotechnology, a Guest Editor for the IEEE Trans. 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.



Ahmed Louri is the David and Marilyn Karlgaard Endowed Chair Professor of Electrical and Computer Engineering 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 Electrical and Computer Engineering at the University of Arizona. From 2010 to 2013, he served as a program director in the National Science Foundation's (NSF) 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 recently 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 TRANS. 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 is currently the 2022/23 President of the IEEE Nanotechnology Council.