IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 6, 2017, accepted August 8, 2017, date of publication August 21, 2017, date of current version September 19, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2742698

Fault and Error Tolerance in Neural Networks: A Review

CESAR TORRES-HUITZIL[1], (Senior Member, IEEE), AND BERNARD GIRAU[2]
[1]Information Technology Laboratory, CINVESTAV-Tamaulipas, Ciudad Victoria 87130, Mexico
[2]Université de Lorraine and LORIA, 54000 Nancy, France

Corresponding author: Cesar Torres-Huitzil (ctorres@tamps.cinvestav.mx)

This work was supported by Conacyt, Mexico, under Grant 237417.

ABSTRACT Beyond energy, the growing number of defects in physical substrates is becoming another major constraint that affects the design of computing devices and systems. As the underlying semiconductor technologies are getting less and less reliable, the probability that some components of computing devices fail also increases, preventing designers from realizing the full potential benefits of on-chip exascale integration derived from near atomic scale feature dimensions. As the quest for performance confronts permanent and transient faults, device variation, and thermal issues, major breakthroughs in computing efficiency are expected to benefit from unconventional and new models of computation, such as brain-inspired computing. The challenge is then to find not only high-performance and energy-efficient, but also fault-tolerant computing solutions. Neural computing principles remain elusive, yet as source of a promising fault-tolerant computing paradigm. In the quest to fault tolerance can be translated into scalable and reliable computing systems, hardware design itself and/or to use circuits even with faults has further motivated research on neural networks, which are potentially capable of absorbing some degrees of vulnerability based on their natural properties. This paper presents a survey on fault tolerance in neural networks manly focusing on well-established passive techniques to exploit and improve, by design, such potential but limited intrinsic property in neural models, particularly for feedforward neural networks. First, fundamental concepts and background on fault tolerance are introduced. Then, we review fault types, models, and measures used to evaluate performance and provide a taxonomy of the main techniques to enhance the intrinsic properties of some neural models, based on the principles and mechanisms that they exploit to achieve fault tolerance passively. For completeness, we briefly review some representative works on active fault tolerance in neural networks. We present some key challenges that remain to be overcome and conclude with an outlook for this field.

INDEX TERMS Fault tolerance, neural networks, redundancy, fault masking, fault models, taxonomy.

I. INTRODUCTION

Artificial neural networks models have attracted intensive research interest and enjoyed significant renewed growth in artificial intelligence related applications over the last two decades, e.g., deep learning models based on a feedforward deep network or multilayer perceptron [1]. Indeed for some applications that extract data from the noisy physical environment, speech recognition and visual object recognition, they appear to be the preferable choice. In neural networks research, one of the main problems that has been addressed is the architecture optimization, which aims at appropriately choosing the neural architecture and its parameters for high generalization performance at solving a given task. However, the fact that performance maximization is of primary concern does not necessarily imply that it is the only goal that has been or should be pursued [2]. Artificial neural networks are generally assumed to acquire some other desirable intrinsic features of biological systems such as their tolerance against imprecision, uncertainty, and faults [3], which also make them harder to study or design [4].

According to neurobiological studies, the human brain is able to tolerate a small amount of synapse or neuron faults, or even use noise as a source of computation [5]. Nervous systems are complex, highly massive parallel information processing architectures made of seemingly imperfect and slow, but exceptionally adaptive and power-efficient components that carry out information processing functions [6], [7]. Moreover, brains have the capability to relearn by growth of

new neurons and/or neural connections and/or retraining of the existing neural architecture [8]. Derived from these observations, it is commonly claimed that the majority of neural network models, abstracted from biological ones, have built-in or intrinsic fault tolerance properties due to their parallel and distributed structure, and the fact that usually they contain more neurons or processing elements than the necessary to solve a given problem, i.e., some natural redundancy due to overprovisioning. However, claiming such an equivalent fault tolerance only on the basis of rough architectural similarities therefore cannot hold true in general, especially for small size neural networks [9], [10]. Furthermore, the assessment of fault tolerance across different neural models still remains difficult to generalize, due to fault tolerance is network- and application- dependent, an inconsistent use of the principal concepts exists, and the lack of systematic methods and tools for evaluation across neural models.

Computational studies have shown that neural networks are robust to noisy inputs and they also provide graceful degradation due to their resilience to inexact computations when implemented in a physical substrate. The tolerance to approximation, for instance, can be leveraged for substantial performance and energy gains through the design of custom low-precision neural accelerators that operate on sensory input streams [11]–[13]. However, in practice, a neural network has a very limited fault tolerance capability and, as a matter of fact, neural networks cannot be considered intrinsically fault tolerant, without a proper design. Furthermore, as a consequence of computation and information are naturally distributed in neural networks, error confinement and replication techniques, key to conventional fault tolerance solutions, cannot be applied directly so as to limit the error propagation when implemented in potentially faulty substrates.

Obtaining truly fault tolerant neural networks is still a very attractive and important issue to obtain more biological plausible models, both for i) artificial intelligence based solutions, where, for instance, pervasive embedded systems will require smart objects fully merged with the environment in which they are deployed to cope with unforeseeable conditions [14]–[16], and ii) as a source to build reliable computing systems from unreliable components, as suggested by [17]. Rooted on the neural paradigm computing systems might take advantage of new emerging devices at nanoscale dimensions and deal both with manufacturing defects and transient faults as well [18], [19] and even considers faults/errors an essential and intrinsic part of the design.

In this last direction, the robustness and the potential fault-tolerant properties of neural models call for attention as permanent and transient faults, device variation, thermal issues, and aging will force designers to abandon current assumptions that transistors, wires, and other circuit elements will function perfectly over the entire lifetime of a computing system, relying mainly on digital integrated circuits [20]–[23]. To achieve real benefits from future technologies at nanoscale, we must find inexpensive ways to exploit such imperfect components from the beginning or even use components whose functionality degrades with time without compromising functionality. As a consequence, computational organizations must be prepared for faults/errors, and provisioned to be able to exploit late-bound information about how variation and faults are affecting the system over time [24]. More specifically, from a pragmatic point of view, the potential fault-tolerant property of neural models will be crucial to the success of attempts to integrate large neural models onto silicon for embedded applications, when problems of yield become unavoidable [25], [26]. Custom hardware implementations of neural networks can benefit the emerging high-performance machine learning applications but faults can compromise the reliability of such accelerators under nanoscale manufacturing process in practical scenarios.

Fault tolerance in a conventional digital computing system is usually achieved by increasing its redundancy in space, time or code, [27], [28] combined with some sort of centralized voting-based strategies, which usually implies higher implementation costs and lower performance that sometimes make it even infeasible to be applied in computing systems at large scale. Research around fault tolerance capabilities of neural networks is expected to provide novel solutions to improve existing fault tolerance and reliability technologies and play a more fundamental role in the future. The style of neural computation, the parallel, and distributed architecture of neural models have been argued as the source for inherent fault tolerance but more general and comprehensive analysis for large class of perturbations affecting neural computation, and large scale fault tolerance mechanisms tailored to neural models must be envisioned at an affordable cost by further exploiting the inherent capabilities of neural computing [29], [30]. As such, a literature review is important to understand how fault/error tolerance in neural networks has been addressed and to gain insight in the foundations and recent developments in this field towards new promising directions. This survey is of great value to investigate how faults/errors will affect the operation of hardware neural networks and whether the faults/errors can be mitigated by leveraging the intrinsic features of neural networks with complementary techniques.

In the literature, several experimental and less analytic works have been carried out to study neural networks fault tolerance related issues, which include the analysis on effect of noise on the output sensitivity [31], [32], the weight error sensitivity [33]–[35], and the relationship among fault tolerance, generalization and model complexity [2], [10], [36]–[38]. Such works have been carried out at different levels of abstraction, from very specific low level physical implementations to the high level intrinsic fault masking capacity of neural paradigms. In fact, most works use a high level approach focusing on errors instead of faults. Despite of an important number of works for fault tolerance in neural networks exist, a survey providing a framework for fault tolerance study and a categorization for the discussion of

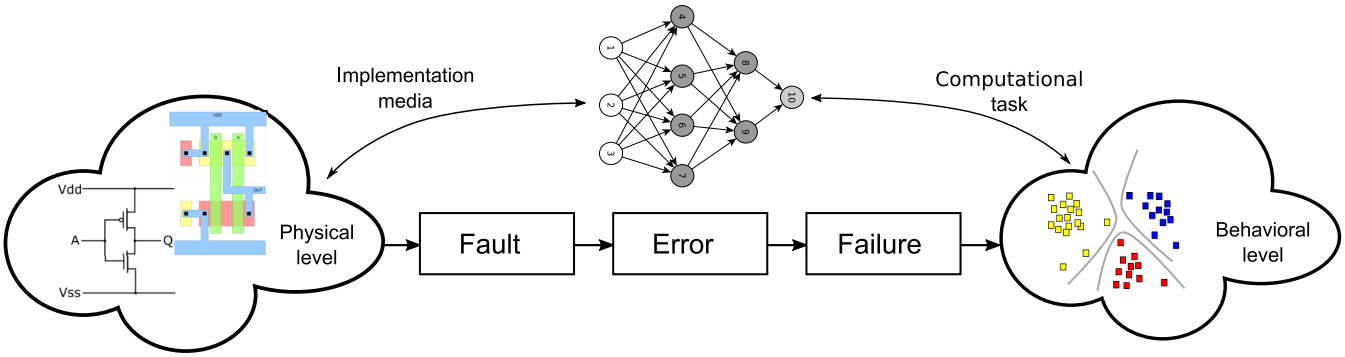

FIGURE 1. Cause effect relationship between fault, error and failure, and its propagation from the physical-implementation level to the behavioral application level of a neural network model.

formal techniques and methods that produce fault tolerant neural networks is still missing.

In this paper, a review on reported works addressing the fault tolerance of neural networks for a given behavioral fault/error model, which evaluate the impact of such errors on the neural computation in a rather technology-independent way, is presented. This paper proposes a categorizing framework that groups together a number of existing techniques to improve fault tolerance into categories and compare their advantages and drawbacks. The rest of this paper is organized as follows. Section II presents background and some key concepts for fault tolerance in neural networks and discusses the similarities and differences between them. Section III formalizes fault tolerant neural networks and presents typical fault models and measures that have been used for fault tolerance assessment. Section IV presents the taxonomy and a discussion of the principal techniques that have been used to produce fault tolerant neural networks, focusing on passive fault tolerance. We present and discuss commonly cited techniques for each class in the taxonomy. In section V we describe some open challenges for current/future research. Section VI provides some concluding remarks.

II. BACKGROUND AND TERMINOLOGY

Neural networks are claimed to have a built-in or intrinsic fault tolerance property mainly due to their distributed connectionist structure. Fault tolerance in a neural network is directly related to the redundancy introduced because of *spare capacity* (over-provisioning), i.e., when the complexity of the problem is less than the raw computational capacity that actually the network can provide [39]. Nevertheless, the analysis and evaluation of fault tolerance remain difficult because many different architectural and functional features under diverse conceptual frameworks are usually involved, and there are no common systematic methods or tools for evaluation [40], [41]. Technical and quantitative reasoning about these features calls for clear definitions, highlighting their similarities and differences, as those concepts appear in different contexts and areas of application.

This section provides some basic definitions related to faults, fault models, fault tolerance and other alike terms, which are widely used in computing systems at hardware level and that have been also applied and extended in neural computing. The interested reader is referred to [42]–[45] for further information on fault-tolerant systems, concepts and principles.

A. FAULT TYPES

There are three fundamental concepts in fault-tolerant systems, which are fault, error, and failure. A cause-effect relationship exists between them, from the physical level to the behavioral level, as conceptually shown in figure 1 for a neural network that performs a computational task and is implemented in a digital substrate.

A *fault* is an anomalous physical condition in a system that gives rise to an error. An *error* is a manifestation of a fault in a system, the deviation from the expected output, in which the logical state of an element differs from its intended value [43], [46]. A *failure* refers to a system's inability to perform its intended functionality or behavior because of errors in its elements or perturbations in its environment. Propagation of an error to the system level results in system failure, however, a fault in a system does not necessary result in an error or failure as it might go inactivated. A fault is said to be active when it produces an error; otherwise it is called dormant.

Faults can be classified by their temporal characteristics as follows:

- A **permanent** fault is continuous and stable with time; it is mainly a result of an irreversible physical damage.
- A **transient** fault may only persist for a short period of time and it is often result of external disturbances.

Transient faults, which recur with some frequency are called **intermittent**. Usually, an intermittent fault results from marginal or unstable device operation and they are more difficult to detect than permanent ones. Transient and intermittent faults cover the vast majority of faults which occur in digital computing systems built with the current semiconductor technology [47]–[49]. Even, future implementation technologies are expected to suffer transient faults due to a reduced device quality, exhibiting a high level of process and environmental variations as well as considerable performance degradation due to the potential high stress of materials [50].

Timing faults change the timing behavior rather than the structure of circuits; they affect circuit parameters which define the timing characteristics of the device, such as propagation delay, hold and set-up times, etc.

Figure 2 shows a classification of some fault types according to their temporal characteristics, indicating some typical causes and mechanisms that generate them, which are modeled with the corresponding permanent/transient fault models, shown as gray rounded boxes in the figure. For instance, a bridging fault occurs when two leads in a logic network are connected accidentally and wired logic is performed at the connection [51]. There exist other fault classifications using different criteria, such as value and extent as proposed in [42].

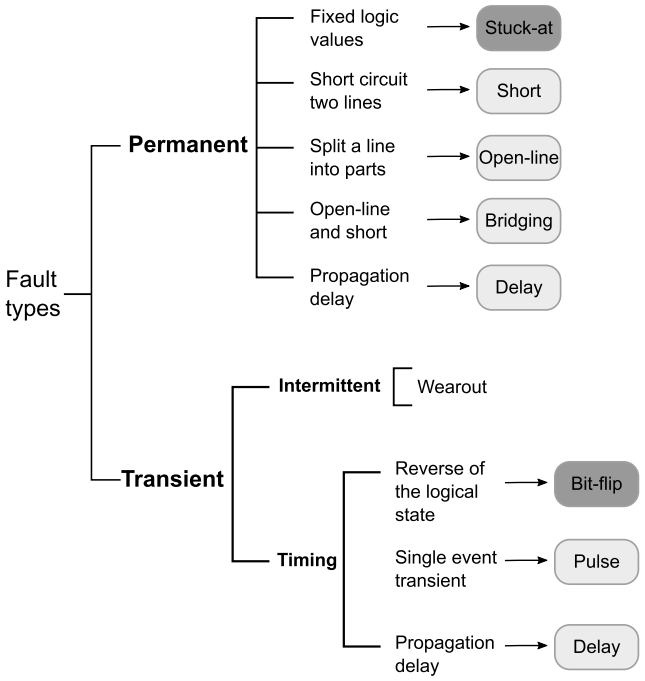

FIGURE 2. Fault types with some representative causes and mechanisms of permanent and transient faults and the corresponding fault models, shown as gray rounded boxes.

In order to facilitate the detection of faults and the correction of their errors, researchers develop models of them to examine the variety of faults that need to be tolerated during the operation of a given system.

B. Fault Models

A fault model lists which components can become defective in a system, and also when and how they will misbehave. They describe the physical manifestation of faults, the types of faults and where and how they will occur in a system [52]. The two major requirements for defining fault models, which group faults that cause similar effects into the system, in some sense, are contradictory. On one hand, *accuracy* is pursued, that is, realistic faults should be modeled and on the other hand *tractability*, complex or large scale systems should be studied at affordable computational costs. Research, therefore, deals with deriving realistic models at higher levels of abstraction, which can accurately capture the faults at lower physical levels.

The following fault models have been widely, and successfully used as abstractions of physical defect mechanisms in digital electronics devices and systems [53]–[55]:

- **Stuck-at**, a data or control line appears to be held exclusively high (stuck-at-1) or low (stuck-at-0).
- **Random bit flips**, a data or memory element has some incorrect, but random value.

The stuck-at fault model has been the source of a great research effort in fault tolerance. It is still very popular since it has been shown that many defects at the transistor and interconnection structures can be modeled, as permanent faults, at the logic level with reasonable accuracy [21], [56]. The stuck-at model is a binary model that do not capture the indeterminate states that faults may induce while occurring. Also, but less frequently for fault tolerance assessment in computing systems, stuck-open or stuck-short faults are considered in the literature [28]. Stuck-open models are necessary, for instance, to characterize the fact that a floating line has a high capacitance and retains its charge for a significant length of time in current semiconductor technology.

The random bit-flip model is intended to model transient faults that usually happen at registers or memory elements due to external perturbations, for instance, a single event upset. Under this model, damage/corruption is done only to the data and not to the circuit itself. Conceptually, it consists of a register bit that is switched randomly, resulting in that memory element holding a wrong logic value. The related pulse model accounts for bit flips produced in combinational logic is used to differentiate from the bit-flip produced in memory circuits. Recall that, single event effects (SEE) in microelectronics are mainly caused when highly energetic particles, present in the natural space environment, strike sensitive regions of a microelectronic circuit [57], however, it is expected that those effects happen in normal environments due to near atomic scale integration as well.

As for fault-tolerant hardware implementations, high level fault models should be consistent with manufacturing defects or physical ones. Indeed, recently, in spite of its importance, it has been shown that classical stuck-at and bit-flip fault models are not enough to cope with the fault mechanisms of new deep-submicrometer technologies and new fault models are needed to cover aspects like transient pulses, indetermination, delays, stuck-opens, shorts, open-lines, and bridgings [54], [55], [58], [59], some of which are illustrated in figure 2 with their corresponding fault models in gray rounded boxes.

On the other hand, contrary to fault models, *error models* do not attempt to capture or locate the underlying physical effect of a failure [28]. They rather characterize the deviation, due to the fault, of the function performed from input to output within a system, at a higher level of abstraction for a better tractability. Mapping criteria of physical faults onto the abstract errors are required to show the usability and consistency of the error analysis in evaluating the actual fault tolerance of a system physical implementation.

Yet, similar errors can be induced by different types of faults, as no one-to-one correspondence might exist.

C. FAULT TOLERANCE AND RELATED TERMS

The main goal of this section is to identify the proper subset of concepts, and highlight their intersection, to develop a common and consistent understanding of their meaning without reference to a specific discipline or implementation media, and then use them with the exposed in the previous subsection concepts for an analytical framework that can be used for fault tolerance in neural networks. Some terms of interest related to the dependability or truthwordiness of a system [42], [60], [61] are as follows:

- **Reliability**, a system is reliable if it performs correctly with high *probability* in the presence of faults under previous stated conditions and for a specified period of *time*.
- **Fault tolerance** is the property that guarantees the proper operation of the system in the event of fault(s) within some of its components.
- **Graceful degradation** is referred to as a low sensitivity to the occurring faults instead of a complete or catastrophic failure.
- **Robustness** is a property that allows a system to continue operating correctly despite noise in its inputs or parameters variation.
- **Error resilience**, tolerance to inexact or approximate computations as originally designed.

Reliability is a quality over time and it is associated with unexpected failures of systems. Understanding why these failures occur is key to improve the system performance in specific working environments. Reliability is a measure of uncertainty and therefore estimating reliability means using statistics and probability theory.

Fault tolerance is often associated with robustness to noisy inputs, functioning correctly in the presence of such inputs, but they are rather different terms [61]. Fault tolerance might generally exploit some sort of redundancy to provide the functionality needed to counterbalance the effects of faults. The redundancy might be manifested mainly in two ways: *extra time* or *extra components* [28].

Intuitively, the term graceful degradation means that a system tolerates failures by reducing its functionality or performance, rather than going into a catastrophic behavior. In order to graceful degradation be possible, the system must have some level of reduced or auxiliary functionality; i.e., it must be possible to define the system's state as *working* but not completely functional.

Error resilience of systems means that they tolerate some accuracy reduction, or inexact computations [62], in return for potential resource savings. Approximate computing exploits the gap between the level of accuracy required by applications and that provided by the computing system, for achieving diverse optimizations; it is more related to specific implementations [11]. On the other hand, it can be said that a robust system provides a graceful loss in performance accuracy when perturbations (e.g. noise) affect its parameters. Hence, a system might both be resilient to lower accuracy (i.e., reduced number of bits) and tolerant to a class of parameter fluctuations or perturbations [2].

According to the concepts exposed above, fault tolerance can be defined as the attribute of a system that allows it to preserve its expected behavior after faults have manifested themselves within the system [42]. More preciselly, for the purpose of this review, a fault-tolerant system might be defined as one that has provisions to avoid failure, as measured by a figure of merit, after faults have caused errors within the system.

D. ACTIVE AND PASSIVE FAULT TOLERANCE

Fault tolerance can be classified into *passive* and *active*, taking into account the mechanisms by which it is achieved in a system. A system with passive fault tolerance does not react in any special way to compensate for the effect of internal faults, but by exploiting the intrinsic redundancy and fault masking, built into the system structure, which efficiently masks the fault effects ensuring correct outputs in spite of such faults [63]. The system is designed to mask, by compensation, a given maximum number of faults. No diagnostics, relearning, or reconfiguration is needed in such passive fault-tolerant system. Thus, fault detection and location can be totally avoided under this approach.

On the other hand, a system with active fault tolerance, explicitly and dynamically recognizes and manages its redundant resources to compensate the fault effects (by adaptation, retraining or self-repairing mechanisms) when they appear. Active fault tolerance requires special detection/localization and supervising/control components, whose design may turn out to be rather complex and intrusive [64]. Active fault tolerance provides a system the ability to recover from faults by reallocating the tasks performed by the faulty elements to the fault-free ones [65].

Generally speaking, it is more difficult to achieve the same degree of fault tolerance of an active approach than with the passive approach, mainly because not all the faulty scenarios can be considered at the system design, and no repair or reconfiguration is possible afterwards. However, in a *hybrid* approach, passive and active tolerance can complement one to each other; a static base configuration masks a given number of faults, while faulty modules are detected online and replaced within fault-free ones in the base configuration [43].

III. FAULT-TOLERANT NEURAL NETWORKS

Since a neural network relies on its neurons to collectively perform its function, a claimed property of neural networks is that they can still perform their overall function even if some of the neurons/synapses are not functioning. Neural networks are not commonly built with the exact or minimum number of neurons to perform a computation for solving a given task. In fact, it has been experimentally observed and documented that such overprovisioning leads to a natural robustness and potential fault tolerance, considering a neural network as a fully parallel and distributed system where neurons/synapses

can fail independently. Yet, the relationship between overprovisioning and the actual number of tolerated faults has not been fully investigated [39].

A. A BASIC DEFINITION

A neural network $\mathcal{N}$ performing a computation $\mathcal{H}_{\mathcal{N}}$ is said to be fault tolerant if the computation $\mathcal{H}_{\mathcal{N}_{fault}}$, performed by a faulty network $\mathcal{N}_{fault}$ obtained from $\mathcal{N}$, is *close* to $\mathcal{H}_{\mathcal{N}}$. Formally, for $\epsilon > 0$, $\mathcal{N}$ is called ϵ-*fault-tolerant*, [39], [66], if it tolerates faulty components (for instance neurons/synapses) for any subset of size at most n_{fails}:

$$\left\| \mathcal{H}_{\mathcal{N}}(\mathcal{X}) - \mathcal{H}_{\mathcal{N}_{fault}}(\mathcal{X}) \right\| \leq \epsilon, \quad \forall \mathcal{X} \in \mathcal{T} \tag{1}$$

where $\mathcal{X}$ is any stimuli, applied to the networks $\mathcal{N}$ and $\mathcal{N}_{fault}$, that belongs to the training set $\mathcal{T}$ or is part of the input data to be processed by the networks. Given a problem, he goal for fault tolerance is to determine the network $\mathcal{N}$ that performs the required computation and has the additional property that is ϵ-fault-tolerant with respect to $\mathcal{T}$.

Furthermore, in a strict sense, a neural network is truly or complete fault tolerant with respect to a class and number of faults if their effects measured by the chosen figure of merit is null. The complete fault tolerance requirement can be weakened toward graceful degradation if we allow that the increase in the error is below a predefined threshold as stated in equation 1. Thus, recall that, when a statement about fault tolerance is made, it should be implicitly assumed a failure condition or criterion of the network functionality, which is the threshold below which it cannot longer perform its function according to the specification. As such, fault tolerance in neural networks depends on the definition of the acceptable degree of performance and its intended application [67].

B. FAULTS IN NEURAL MODELS

At a high level of abstraction, fault tolerance, within neural models, might be analyzed by the effects of errors in the main operators that support the whole neural computational task, rather independent from their intended physical implementation. In fact, this has been the practice in most works reported in the literature. In a more comprehensive and structured approach, as the one described in [29], after this initial step, physical faults affecting a specific implementation can be mapped onto such errors so that the expected fault tolerance of a given architectural implementation of a neural model can be estimated, and further by identifying critical neural components, complementary and ad-hoc fault tolerance policies can be further applied to enhance the properties of the neural model implementation.

In neural networks, an error model can be defined depending only on the neuron behavior itself, rather independent of its physical implementation, which is usually targeted to a digital substrate, so as to estimate the influence of faults on the neural computation from the initial design stages. More specifically, in the behavioral neuron model, as conceptually shown in figure 3, errors may occur [68], [69]:

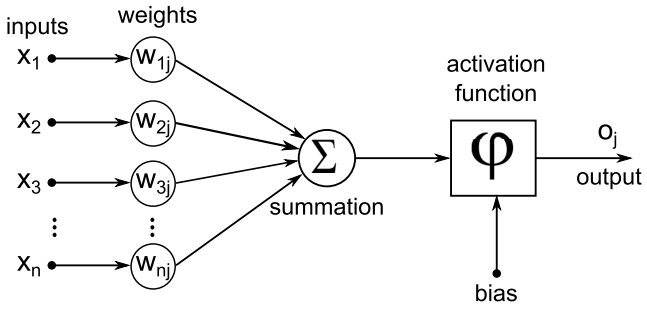

FIGURE 3. Abstract neuron model and its main components.

- As unexpected values of signals in the communication channels due to faulty interconnections or noise.
- In the synaptic weight or the associated computation, which in the absence of implementation details can be considered as indistinguishable.
- In the neuron body itself, affecting the summation or the evaluation of the nonlinear activation function.

The first two errors, in digital implementations, are often modeled as both stuck-at-0 or stuck-at-1, since an asymmetric behavior for such faults has been reported [70]. Synapse errors are modeled at stuck-at-*value*, where *value* is within the domain for weights $[w_{min}, w_{max}]$. Errors caused by faults in the neuron body, usually saturates its output to positive/negative values, thus they are modeled as stuck-at-1 or stuck-at-(-1), as the activation function is often in this range. However, neurons cannot only stop computing by saturating, but generally they might even send a value different from their nominal expected output of the transfer function [65]. Neurons that can fail by transmitting arbitrary values (known in the literature as Byzantine neurons) has been only recently considered for fault tolerance assessment [39].

The stuck-at model essentially allows to investigate fault tolerance at the behavioral level, independently of the actual implementation or detailed characteristics of physical faults. It abstracts and simplifies faults into stuck-at values affecting single components. Such an abstraction has been widely used in testing of digital circuits and has proved to be sufficient to model a large number of physical faults. Some other faults/errors can be even considered for neurons but they can mask each other in the sense that it can be undistinguishable which fault occurred, for instance a fault in the synaptic operation itself (multiplication) instead of a fault in the weight storage. Considerations on physically realistic fault models for analog VLSI neural networks are also needed [71]–[73].

Among the most important works reported in the literature regarding fault models in neural networks, mostly feedforward multilayer networks, are the following.

Sequin and Clay [68] used a bottom-up approach to categorize the types of faults that usually might occur in neural networks looking at the main components that comprise a network and focusing in fault cases that yield a worse effect on the overall performance of the network. In their modeling, authors distinguished three types of units, input, output and

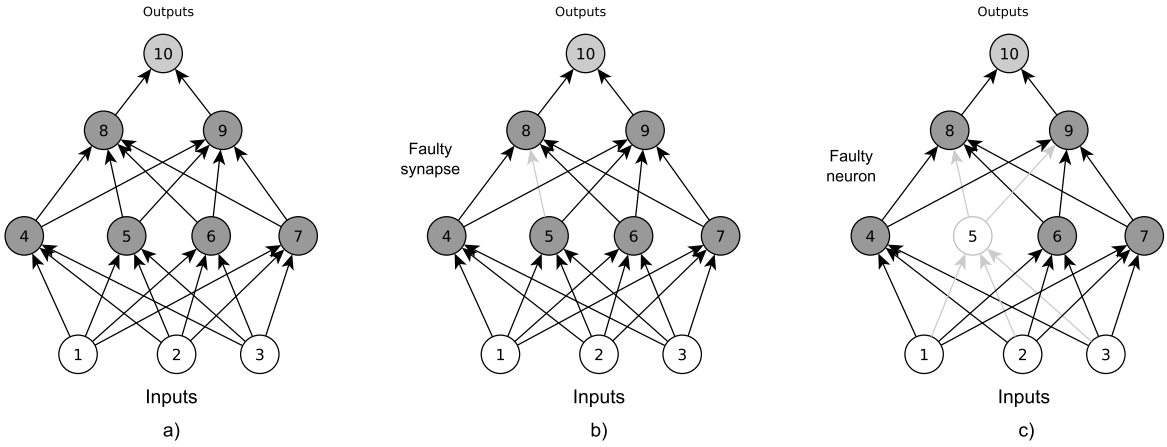

FIGURE 4. a) A feedforward neural network, b) A faulty synapse between node 5 and 8, c) Network considering that node 5 is faulty.

hidden units, all of which can fail and potentially impact differently the network operation because of their location. They focused in the following types of faults: i) missing hidden units stuck at an intermediate output value, regardless of its inputs, not delivering an effective signal, ii) saturated hidden units stuck at an extreme value, iii) missing weights, so called disabled, which do not transmit any signal, and iv) saturated weights, the weight is driven to the maximum or minimum values of their allowed range.

Bolt [52] introduced a method to develop fault models for neural networks at the abstract level, considering the fault location and then defining the faults characteristics, by enumerating the manifestations of a fault to be such that the maximum harm is caused to the neural network. A fault model for the multi-layer perceptron was developed, at a high level of abstraction, thus allowing their inherent fault tolerance to be estimated. Two types of fault components were identified, stable entities whose associated information does not change at any time, such as weights and activation functions, and temporary entities whose information is only valid for a limited period of time, such as outputs and activation values.

Chandra and Singh [8] investigated pre-trained feedforward neural networks and proposed a framework of study for neural fault tolerance. Particularly, they proposed fault models and fault/error measures to quantitatively assess fault tolerance in such feedforward networks. According to their proposal, fault tolerance can be divided intro three separate sets of categories: i) tolerance to faults/errors in the learning rule, ii) tolerance to faults/errors outside the neural network structure (incorrectness in the inputs due to noise), and iii) tolerance to fault/errors inside the neural network (structural fault). Specific faults were defined according to such proposed categories.

C. FAULT INJECTION AND MEASURES

For fault tolerance assessment, a fault injection method is required for gaining insights of the behavior of a system [74]. Especially, its critical components might be identified, which can then be protected against possible faults targeted to a specific physical implementation. Faults are probabilistically introduced into a neural model and the degree of failure, impact on the performed neural computation, is evaluated according to some measures. Figure 4 shows an example of a feedforward neural network, and the corresponding derived networks when a faulty synapse/neuron is considered in the connection graph. The measure of fault tolerance from many experiments can be evaluated against the number of considered faults injected into the neural model. The limit of the fault tolerance of the network, assessed in this way, is problem-dependent and is determined by operating scenarios of multiple faults that would lead to a violation of the performance constraints. With known failure rates and faults occurring at random locations, these worst-case scenarios can be used to estimate an upper bound for the fault tolerance of the neural network [67]. If a minimum number of faults is stablished, n_{fails}, it is necessary to prove that the network will perform well with n_{fails} or fewer faults from a specified set.

For large neural networks, exhaustive testing of all possible single faults is prohibitive, not to mention that even multiple faults might occur concurrently. Hence, the strategy of randomly testing a small fraction of the total number of possible faults in a network has been adopted for tractability. It yields partial fault tolerance estimates that statistically are very close to those obtained by exhaustive testing. Moreover, when the fraction of faulty components tested is held fixed, the accuracy of the estimate generated by random testing is seen to improve as the network size grows [65].

Table 1 summarizes some general measures used to assess fault tolerance, which basically measure the performance distance (closeness) between fault-free or a baseline neural network and the derived faulty networks in classification tasks. The measures selection is problem and neural model dependent but broadly fall into two main categories of the neural paradigm: those requiring supervised or unsupervised learning. Chandra and Singh [8] suggested the use of

TABLE 1. Some typical measures used to assess performance and fault tolerance in neural networks.

Measure	Expression	Comments
Supervised learning		
Mean squared error (MSE)	$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$	Average the square prediction error for multiple instances
Mean absolute percentage error (MAPE)	$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left\lvert\frac{Y_i - \hat{Y}_i}{Y_i}\right\rvert$	Average the absolute error in the predictions
Network sensitivity (S)	$S = max(overfault\left(\frac{x}{y}\frac{\Delta y}{\Delta x}\right))$	Measure the change in the output due to the change that is produced in the input or a parameter
Memory capacity (MC)	$MC = \frac{\min(n,p)}{2\log\min(n,p)}$	The number of bits of information that can be stored
Unsupervised learning		
Silhouette coefficient	$sw_i = \frac{(b_i - a_i)}{max(a_i,b_i)}$ $s\bar{w} = \frac{1}{n}\sum_{i=1}^{n} sw_i$	Give a measure of the quality of the clusters obtained

mean squared error (MSE) and the mean absolute percentage error (MAPE) to measure the effect of faults, and particularly for classification problems, the percentage of misclassification is suggested. For other MSE-like and sensitivity related measures see [37], [76], [77]. On the other hand, sensitivity measures the change in the output due to a change in the input or internal parameters [69]. The memory capacity has been used for evaluating fault tolerance in associative memories with faulty neurons [78].

For neural models for clustering tasks, silhouette statistics can be used as a fault tolerance measure since ground truth labels are not known. It gives a measure of the quality of the clusters obtained. Such measure is defined for each sample and is composed of two scores: i) a_i, the mean distance between a sample and all other points in the same cluster, and ii) b_i, the mean distance between a sample and all other points in the next nearest cluster. The silhouette coefficient score is bounded between -1, for incorrect clustering, and $+1$, for highly dense clustering. Scores around zero indicate overlapping clusters.

As a matter of fact, such figures of merit measures two overlapping aspects, on one hand how well the problem is solved and on the other hand the fault tolerance that the corresponding network would roughly provide. But those measures do not provide a more comprehensive fault tolerance assessment, such as for example, on the tight bounds on the number of neurons that can fail, without harming the result of a computation in terms of weight and failure distribution [39]. This call for new measures for better understanding of the extent to which neural networks can be fault-tolerant.

IV. TAXONOMY OF FAULT TOLERANCE

Different starting points and criteria usually might lead to different taxonomies of fault tolerance. A general but widely adopted frame is to classify fault tolerance as *passive* or *active*, based on the principles and mechanisms that they exploit to achieve fault tolerance as outlined in section II-D, and particularly emphasized for fault tolerance in neural models in [67]. We follow this frame in reviewing the literature related to neural networks fault tolerance, and we principally focus on methods and techniques to enhance fault tolerance passively. Nonetheless, other important works on fault tolerance in neural networks are also briefly referred throughout this review. As for instance, the work reported in [79], where an empirical study of the influence of the activation function on fault tolerance properties of feedforward neural networks is presented, showing that the activation function largely has relevance on fault tolerance and the generalization property of the network. Furthermore, for completeness in section IV-B, we briefly review some representative works on active fault tolerance in neural networks.

Before going into details, it is worth to point out that the majority of reported works has been focused in feedforward neural networks and few attempts have been made to improve fault tolerance in some other neural models. In section IV-C works that discuss and analyze fault tolerance of non-feedforward neural networks will be briefly described, even though some works do not propose any specific technique for fault tolerance improvement. This issue is of importance since the studies and results in the literature concerned with fault tolerance in feedforward neural networks, despite of its importance (e.g. for deep learning), are difficult to generalize and directly apply across other different neural models.

A. PASSIVE FAULT TOLERANCE

In the proposed taxonomy, as schematically shown in figure 5, the reviewed works are classified based on their main strategies to achieve or improve fault tolerance in the recall stage of neural networks without considering retraining, i.e., we mainly focus on passive fault tolerance. Since only passive fault tolerance is considered in depth herein, the main mechanisms to provide the needed redundancy or fault masking to enhance fault tolerance will be presented. Each technique is explained based on its characteristics, design objectives, and the considered fault types in the performed study.

Three main categories, in the passive approach, are identified, which group together related methods and techniques to enhance the intrinsic fault tolerance capabilities of neural networks: i) explicitly augmenting redundancy, ii) modifying learning/training algorithms, and iii) neural network optimization with constraints.

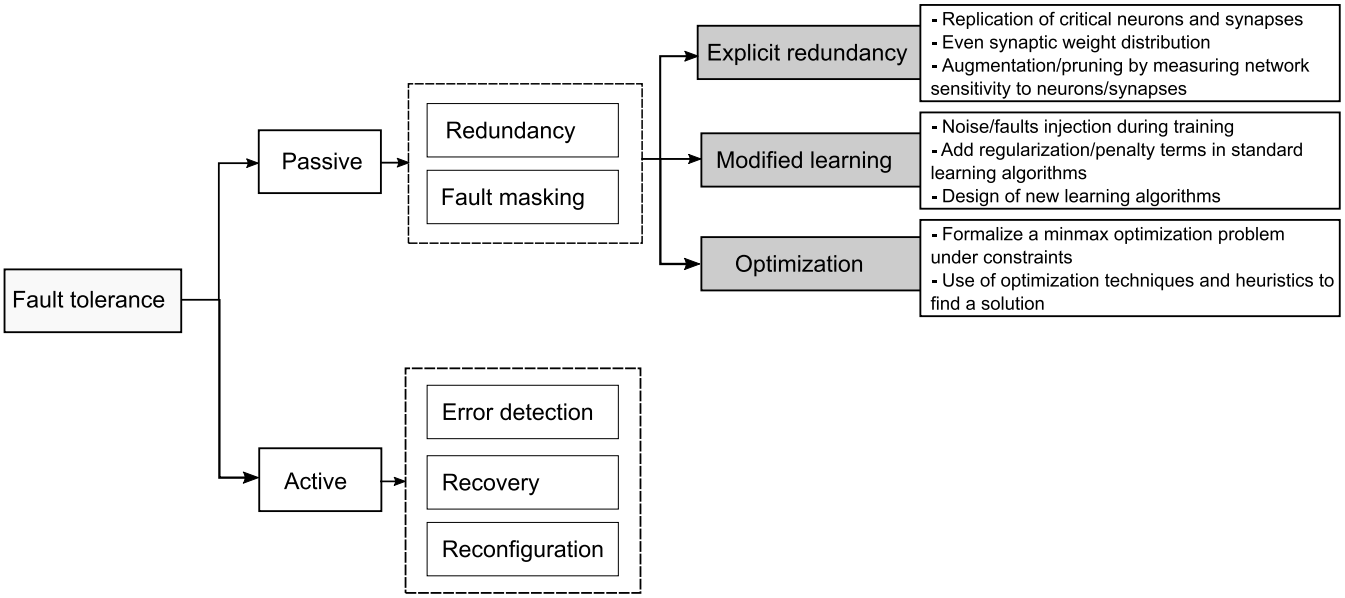

FIGURE 5. Taxonomy for techniques and methods to enhance fault tolerance in neural network models grouped into two main categories, passive and active fault tolerance. This survey mostly reviews research on the three subcategories of passive fault tolerance.

TABLE 2. Summary of some representative works for enhancing fault tolerance in neural models by explicitly adding redundancy in the network after training, with a representative example of a NN topology used in the experiments.

Reference	Fault model	NN topology	Comment
[80]	Stuck-at	100-200-14	Redundancy applied on critical neurons within the so called isomorphic sets (neurons having identical inputs)
[81]	Stuck-at	259-20-6	Replication of all the hidden neurons and their associated connections in the network
[82]	Weight perturbation	4-10-3	Removing/replication of nodes based on the sensitivity of links/nodes in the network output
[83]	Stuck-at	2-20-4	A hybrid approach that add new nodes that share the load of the most critical nodes in the network
[65]	Stuck-at	60-42-1	Exploit the intrinsic weighted summation performed by neurons to replicate the hidden units
[84]	Stuck-at	3-4-1	Identify potential critical faulty elements and duplicate inputs, bias, weights or neurons, according to the evaluation criteria

1) EXPLICITLY AUGMENTING REDUNDANCY

Fault tolerance in neural networks can be achieved by explicitly inserting redundancy in a pre-trained network, particularly in hidden neurons and their associated connections within feedforward networks. Such methods start with a minimal network that learns the given input/output pattern mapping or performs the desired computational task, and then replicate hidden neurons or selectively add nodes to share the load of the critical ones, after the training has been completed. The fault tolerance criterion is tested on the obtained model and improved off-line by using techniques such as: network augmentation by means of spatial redundancy, i.e., replication of critical neurons, or evenly synaptic weight distribution with neuron pruning and removal of unnecessary weights. Some representative works in this category are summarized in table 2, indicating the fault model, a typical network size and topology and a short description on the underlying principle, which is further described in the following paragraphs.

Chu and Wah [80] addressed fault tolerance using a hybrid redundancy scheme applied on well-trained multilayer neural networks in a similar way to the conventional redundancy approach. A combination of spatial, temporal and information redundancy was applied on critical neurons, those ones located in the network outputs. Neurons are classified into a limited number of isomorphic sets in which neurons have identical inputs. By replicating weights of one neuron into the others in the same set, it is possible to recompute the output of a given neuron and to cope with transient, intermittent, and permanent faults. Output neurons are implemented as time-multiplexed in synchronous processing elements as they receive, process and send outputs at the same time in the network. A decision automata is used as a centralized supervisor for error detection and recovery but a fault model was not specifically defined in this work. The high level analysis was carried out in terms of the overhead and implementations costs that incur redundant neurons.

Emmerson and Damper [81] investigated fault tolerance of single and multilayer perceptrons (MLPs), especially for pattern recognition tasks. Backpropagation-trained MLPs with different hidden nodes were considered in this work.

After training, networks were subject to random connections cuts, as a physically plausible type of fault. Experiments, repeated several times and averaged, showed counterintuitively that fault tolerance does not improve as the number of hidden units increases, and that backpropagation training fails to exploit redundancy (additional hidden units). They proposed a mechanism called *augmentation* to improve fault tolerance, consisting in the replication of each hidden neuron and their associated connections. Since each node now has twice as many inputs as in the original network, the weights connecting the augmented network's hidden layer to the output layer must be the half of those in the original network to maintain the same input-output mapping, as it is shown in the example in figure 6. Augmented networks showed better fault tolerance, and the inserted redundancy, the excess nodes, was verified by means of singular value decomposition.

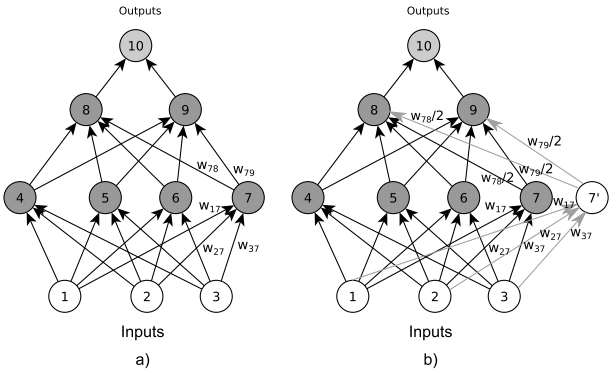

FIGURE 6. a) A critical hidden neuron (7) in a feedforward neural network and b) Explicitly augmenting redundancy by duplicating neuron 7. The postsynaptic weights of neurons 7 and 7′ are halved.

Chiu *et al.* [82] addressed fault tolerance of feedforward neural networks by measuring the sensitivity of links and nodes in the network output, and implemented a technique to ensure the design of networks that satisfy well-defined fault tolerance criteria. Their method takes as input a well-trained network and then follow two main steps, i) unimportant nodes in the hidden layers are removed, according to the sensitivity measure and a threshold, and ii) the pruned network is retrained and some redundant nodes are introduced to this network so as to share the task of the critical nodes (neurons with high sensitivity). Faults are injected as a weight perturbation, and sensitivity of links and nodes are evaluated in terms of the MSE. Two criteria for adding nodes are used, 1) adding extra nodes until the sensitivity of the current most critical node is less than some proportion of the sensitivity of the initial most critical node, and 2) adding extra nodes until the number of nodes is equal to the original number of nodes, in order to compare two networks of the same size. Weights in the augmented network are modified in a similar way as in [81]. The obtained results showed a considerable improvement in the fault tolerance (changes affecting weights) of networks trained for two multiclass classification problems.

Chiu *et al.* [83] extended their previous work [82] and, in this contribution, they proposed three methods for improving fault tolerance of feedforward neural networks under a hybrid approach, which involves both modifying training and use of explicit redundancy. In the first method weights are restricted to have low magnitudes during the backpropagation training, since fault tolerance is degraded by the use of high magnitude weights; at the same time, hidden nodes are added dynamically to the network to ensure that the desired performance can be obtained. The second method adds artificial faults to various components (nodes and links) of a network during training since injecting a specific fault during training can produce a network that can tolerate that specific fault very well. Perturbation of weight values and stuck at zero faults were considered for synapses, and stuck at 0/1 faults for nodes. The third method removes nodes that do not significantly affect the network output, and then adds new nodes that share the load of the most critical ones in the network. Note that the first two methods of this work can be also considered in the second category of the taxonomy for fault tolerance as training was modified.

Phatak and Koren [65] studied fault tolerance in feedforward neural networks with a single hidden layer considering permanent stuck-at type faults of single components. They proposed a method to synthesize fault tolerant neural networks by replication of the hidden units. The method exploits the computational characteristics of the intrinsic weighted summation performed by neurons. It starts with a near minimal network that learns the given input/output pattern mapping. The hidden neurons are replicated as a whole and inputs/biases of the output neurons are scaled down/up accordingly. There is no majority voter to explicitly mask out the faults. Compared to previous works that use stuck-at-0 permanent faults, herein the fault model was extended to allow permanent stuck-at-$\pm W$ type faults on a single component (weight/bias). Analytical, as well as, extensive simulations showed that a significant amount of application-dependent redundancy is needed to achieve complete fault tolerance, despite the somewhat simple restrictive assumption of single faults. Moreover, authors pointed out, as future extensions, to include modifications of learning algorithms to find weights and biases that optimize fault tolerance as a promising alternative to be further explored.

Dias and Antunes [84] proposed a technique to improve fault tolerance by changing the architecture of feedforward neural networks after training, while maintaining its input-output mapping unchanged. Following a similar approach to previous works in this category, this technique evaluates the elements of the network which are more sensitive to a fault and duplicates inputs, bias, weights or even neurons, according to the evaluation criteria. The fault model, used for weights and inputs, is the stuck-at considering 0, *min* and *max* values. The proposed dividing technique diminishes the importance of the fault by splitting a potential faulty synapses and dividing its original strength accordingly. A complete critical neuron can also be duplicated, including all of its

TABLE 3. Summary of some representative works for enhancing fault tolerance in neural models by modifying learning/training, with a representative example of a NN topology used in the experiments. BP stands for backpropagation, WP for weight perturbation, WC for weight constraint, PT for penalty term, and R for regularization.

Reference	Fault model	Algorithm	NN topology	Comment
[68]	Stuck-at	BP	21-14-10	Representative faults are presented during training, without modifying the learning algorithm
[87]	Stuck-at	BP	21-14-10	Comprehensive presentation of patterns considering specific faulty hidden neurons during training
[88]	Stuck-at	BP + WC	2-6-1	Weights magnitude of neurons in the same layer are forced to be within a limited range
[89]	WP	SD + PT	9-6-2	Penalty terms are included for low average weight saliency
[90]	Stuck-at	BP + PT	54-9-3	Constructive incremental algorithm, only weights whose relevance is less than a threshold are updated
[91]	Stuck-at	BP + PT	60-84-1	Update each weight only as long as the new weight does not exceed a threshold
[75]	WP	BP + R	8-16-2	The added term is related to the mean square error degradation in the presence of weight deviations
[92]	Open-fault	T3	2-40-1	Use a validation set to build the fault curve of a trained network to locate the inflection point
[93]	Stuck-at	Recursive training	4-16-3	Special purpose regularization algorithm that imposes some structure on the network weights
[94]	Stuck-at	ad-hoc	77 nodes	An estimation for the performance of faulty RBF networks that define an objective function
[95]	Stuck-at	ad-hoc	88 nodes	Two learning algorithms, one batch mode and one online mode, for RBF networks

connections to the previous layer and the connections coming from this neuron to the next layer will have half of their previous values in the unmodified network. Interestingly, authors in a further work introduces the Fault Tolerance Simulation and Evaluation Tool that evaluates and assists to improve fault tolerance for neural networks [85]. The tool is composed of three main sub-tools: the Insertion tool, to receive the neural network that were previously trained and prepared; the Evaluator, for evaluation of the fault tolerance and the Improver, for improving the built-in fault tolerance in an integrated environment.

As a summary, methods for explicitly augmenting redundancy in the neural model could be effective, but they often result in large networks with too many hidden nodes and parameters. Thus, pruning, used to determine the relevance or contribution of hidden units and to identify excess units that might be removed to produce a reduced network, is of relevance in this approach. Even though these methods do not appear to be different from the conventional redundancy approach, such as triple modular redundancy schemes, they are different in one major respect. There is no majority voter to explicitly mask out the faults, but faults are masked by exploiting the intrinsic characteristics of neural networks such as the weighted summation and the fact that the hidden-layer nodes operate close to their saturation points. However, most of these techniques in this category make a trained network fault tolerant by replication, similarly to the conventional approach for fault tolerance, whereas the main question for neural computing is about the inherent passive fault tolerance of neural networks, as discussed in [86].

2) MODIFYING TRAINING/LEARNING

These methods modify conventional training/learning schemes used for neural networks models in order to tolerate faults a posteriori, i.e., by explicitly targeting fault tolerance while training/learning to achieve the desired computational task. A summary of some representative works in this category is shown in table 3. According to [86], ANN models may be described by the conceptual relation between two main factors, as established in equation 2:

$$\{\text{ANN model}\} = \{\text{Architecture}\} + \{\textbf{Training/learning Paradigm}\} \quad (2)$$

Following this conceptualization, in this category, two main subcategories can be identified. On one hand, some works have focused on the training experience provided to the network for the development of techniques to obtain fault-tolerant networks by adding noise, perturbations or by direct faults injection during training. On the other hand, some other works focus on the learning rule by including a regularization/penalty term in the performance measure to be improved to indirectly incorporate faults in conventional algorithms such as backpropagation, or by a major adaptation/modification of learning algorithms so as to, for instance, search for weights values that are more equally distributed and avoid saliency.

In the first subcategory, Sequin and Clay [68] showed that fault tolerance can be improved by a suitable training process, where a feedforward neural network is presented with representative faults so as to learn an internal redundant distributed representation. They modified the training procedure such that temporary random faulty hidden units

could be injected. For each pattern presentation, from one to three hidden neurons were randomly selected to be faulty. The resulting internal representation assumes a more spread-out and distributed form that is also more tolerant to faults. From experiments, both for classification and approximation tasks, it was observed that training with only single faults can lead to fault tolerance against multiple faults as well. Another key remark is that weight values induce a sharpening of the transition regions of the sigmoids and thus produce more extreme binary output signals. However, for analog approximation tasks it is more difficult to mask the effect of faults.

Similarly, Arad and El-Amawy in [87] presented an algorithm, derived from the backpropagation algorithm, with built-in measures to promote fault tolerance during training. They demonstrated that feedforward neural networks are able to tolerate any combination of two faulty hidden units even with mixed fault types. They considered a pattern presentation as the execution of the forward pass of the backpropagation algorithm for a particular pattern, assuming certain number of faulty hidden neurons with a relatively higher probability. A comprehensive presentation (CP) of pattern p is defined to be the execution of all desired presentations of the pattern. By varying the CPs parameters the network can be trained to exhibit different fault tolerance degrees and varying learning efficiencies. Each hidden-layer neuron can be assumed faulty with a relatively higher probability in each iteration. They argue that the ability to tolerate various fault types can be associated with an increase in the size of the training set due to the larger number of fault types considered. Moreover, they confirmed that using critical fault types, stuck at the extreme values of the activation function, results in fault tolerance against any single faulty neuron stuck at any value which lies between the two extreme values.

Most works in the second subcategory, in order to cope with faults at recall phase, adds a regularization/penalty term to the training cost function so as to bias the solution toward fault-tolerant networks. Commonly, well-known learning algorithms, such as backpropagation, are modified by introducing such terms in the error function to promote uniform information distribution. Regularization is an essential technique that has been proved to be useful to improve generalization ability of neural networks [96]–[98], and by imposing smoothness constraints on the estimated function, small changes in inputs or parameters produce small changes in the computed outputs. Under this approach, there are two main terms in the objective function as shown in equation 3:

$$E + \lambda J \tag{3}$$

The first term E includes the standard error term used in learning algorithms such as backpropagation. The second term J, is the penalty function, which takes into account the errors that arise due to faults, and λ is the regularization parameter that controls the compromise between the degree of fault tolerance and accuracy. When E is minimized, the error between the target outputs and the faulty network outputs is also minimized. Thus, the addition of these terms forces the search to look for a solution with better fault tolerance. The fault tolerance constraints in the modified training algorithms can be interpreted as imposing regularity conditions on the estimated function by the neural network, such as weight decay for penalizing large weights, by adding noise to the weights, and weight smoothing for uniform information distribution.

Wei *et al.* [88] presented a learning method, derived from the backpropagation algorithm, to improve the fault tolerance in classification tasks. The classical backpropagation algorithm develops nonuniform weights with a few that are critical and many others that are not significant. During training, the weights magnitude is constrained to be within a limited range for evenly information distribution across weights. Authors highlight that as some layers are intrinsically more important than others, they only evaluate that information is evenly distributed across the weights, which are between the same pair of successive layers: if a weight has a relatively high influence degree, its magnitude is constrained not to rise temporarily. Two types of faults were considered: i) node fault of stuck-at node's extreme values and ii) connection faults, which consider setting the relevant weights to zero.

Edwards and Murray [89] proposed a method for enhancing fault tolerance via penalty terms, which are incorporated into the learning rule to optimize the networks for smoothness of the solution towards low average weight saliency and optimally distributed computation. Such method focuses on small weight perturbation fault tolerance. Two penalty terms are introduced, a first order term intuitively linked to the use of weight-noise, particularly multiplicative noise, and also a second order term is proposed using the well established statistical theory of smoothing splines. Neural networks were trained using a simple steepest descent algorithm with an incorporated line search technique to optimize the step size in multilayer perceptron networks. Fault tolerance was assessed using the average value of the error hessian, as this value has been shown to be directly related to the inverse of the fault tolerance of a given network.

Hammadi *et* al. [90] proposed a constructive algorithm for fault tolerant feedforward neural networks, which starts with a single hidden neuron and incrementally adds neurons whenever the network fails to converge. The baseline algorithm is modified to update any weight whose relevance is less than a given threshold, and the weights are updated using the backpropagation algorithm. The relevance of a given weight is defined as the maximum error caused at the primary output by the stuck-at fault of this weight. The algorithm consists of three main stages: training a normal network using backpropagation, training of candidates where only input-to-candidate and candidate-to-output connections are trained, and neuron addition. This process is repeated until the convergence criterion is satisfied or the maximum network size is reached. The main fault type considered was the loss of a connection between two neurons (open fault). The used fault tolerance metric was the percentage of recognized

patterns as function of the percentage of faulty weights in the network. The constructive algorithm was based in a previous work, where in [99] proposed a learning method to enhance fault tolerance ability, which uses the Taylor expansion of the output around fault-free weights, to estimate the relevance of the weights to the output error. In this algorithm, the weight that produces the maximum relevance is decreased.

Cavalieri and Mirabella in [91] proposed an algorithm that updates synaptic weights so as to distribute their absolute values as uniformly as possible in a multilayer perceptron with sigmoidal activation functions, based on the observation that a fault in large weights is critical for the fault tolerance of the network as a whole, particularly for weights in the output layer. The modified backpropagation algorithm updates each weight only as long as the new weight does not exceed a given threshold, which in turn is updated dynamically during the learning phases based on the current weight values. The basic principle of inhibiting large absolute weight values is at the cost of larger network training convergence time. Two kinds of faults were considered, stuck-at-0 and stuck-at-1 and fault tolerance was assessed when multiple faults occur.

Bernier *et* al. [100] and Bernier *et* al. [37] presented an algorithm that tries to maximize fault tolerance in a given network. Such algorithm explicitly adds a new term to the backpropagation learning rule related to the mean square error degradation in the presence of weight deviations. Authors presented a quantitative measure to evaluate the fault tolerance and the noise immunity of a multilayer perceptron. This measure, termed mean squared sensitivity, was derived from an explicit relation between the mean squared error degradation of the multilayer perceptron in the presence of perturbations and the statistical sensitivity. This new term can be considered as a stabilizer that tends to smooth the square error surface with respect to the weight values in order to obtain configurations that are stable against perturbations of their values. The proposed algorithm showed better performance with respect to fault tolerance and similar performance with respect to learning abilities to the conventional backpropagation algorithm.

Zhou *et* al. [92] introduced a method called T3 (Test-Train-Test) in order to exploit the fact that the performance of trained neural networks does not linearly decrease with the increasing of the severity of faults characterized by a fault rate. The proposed method uses a multi-node open fault model where several faulty hidden nodes are concurrently considered. T3 utilizes a validation set to build the fault curve of a trained network, then it heuristically locates the inflection point of the fault curve and repeatedly trains the network according to the corresponding fault rate so that the spatial redundancy is added to the network in a proper manner. Eventually, the function of faulty nodes are undertaken by the additional appended nodes, both the number and the function of the appended nodes are different to those of the faulty nodes. The T3 algorithm was only applied to some feedforward neural networks whose hidden nodes are dynamically appended during training.

Simon [93] modified the recursive training algorithm [101] for the optimal interpolative classification network to include distributed fault tolerance against small weight perturbations from their trained values. Recall that, the optimal interpolative network is a three layer classification network that grows only as many middle layer neurons as necessary to correctly classify the training set [102]. The proposed algorithm attempts to distribute the weights evenly throughout the network to achieve fault tolerance in such a way that both the sum of each row of the weight matrix is equal and the sum of each column of the weight matrix is also equal. The proposed technique can be viewed as a special purpose regularization algorithm as it imposes some structure on the network weights, specifically designed to minimize the effect of particular types of faults.

Xiao *et* al. [94] studied the performance of faulty radial basis function (RBF) networks considering a general node fault model, which includes stuck-at-zero, stuck-at-one, and the stuck-at level, with arbitrary distribution. Authors derived an expression to describe the performance of faulty RBF networks and identify an objective function. With this function, a training algorithm for the general node situation was developed. A mean prediction error (MPE) measure that is able to estimate the test set error of faulty networks is derived. As previous works focused in feedforward networks, it is not straightforward to compare the results for these neural network models. In an extension of this work, in [95] studied how the open weight fault and the multiplicative weight noise degrade the performance of RBF networks, and then developed two learning algorithms, one batch mode and one online mode. The first one produces the optimal weight vector with respect to the average training set error of faulty networks. For the online mode, a cyclic learning scheme, in each training cycle an example is learned exactly once according to a fixed order. From the experiments it was found out that when the RBF nodes increase, the fault-tolerant ability can be further improved.

As a summary, methods and techniques that modify training/learning algorithms often significantly increase the computational cost and slow down the convergence of training/learning, but the performance/fault-tolerance tradeoff is solved during this phase without any external interaction afterwards. They try to avoid that key or critical neural elements appear, i.e., synapses/neurons that being faulty cause a great impact on the function of the network or in other words to evenly distribute information among the network weights. Learning can induce intrinsic fault/error masking ability by forcing neurons to work towards the saturation regions of the nonlinear activation functions, so that even a large variation of the weighted summation affects the neuron output marginally [105]. Generally, fault tolerant neural networks generated by these methods appear to exhibit better generalization than unconstrained solutions, and also it appears that enforcing uniformity in the network is similar to making all hidden units equally relevant in the network.

TABLE 4. Summary of some representative works for enhancing fault tolerance in neural models posed as constrained optimization problem, with a representative example of a NN topology used in the experiments.

Reference	Fault model	Optimization	NN topology	Comment
[66]	Node removal	Quadratic programming	128-4-17	Maximize the number of different single hidden units while finding the weights that minimize the error
[103]	Node removal	Least-squares, gradient-based	60-4-2	The methods proposed here lead to networks that exhibit a partial degree of fault tolerance
[104]	Open fault	Genetic algorithms	60-4-2	A fault-tolerant measure, fitness, promotes the population to evolve good fault tolerance

3) OPTIMIZATION UNDER CONSTRAINTS

In this approach, the training/learning process and fault tolerance are transformed into an optimization problem solved by nonlinear optimization algorithms in order to find the neural network topology and its parameters that perform a given task and fulfill fault tolerance constraints as well [104]. Table 4 summarizes some representative works that fall in this category. The fault tolerance constraints in the optimization approach can be interpreted as imposing regularity conditions on the estimated function by the neural network with respect to the weights values.

Usually, the fault tolerance problem in this category has been formulated as a constrained *minimax* optimization problem where the goal is to minimize the maximum deviation from the desired output for each input in the presence of single unit faults in the neural network model:

$$\min_{W} \max_{i \in V_h} E(W^i) \tag{4}$$

subject to the following constraints:

$$d^l - y^l = 0, \quad \forall l = 1, \ldots, p \tag{5}$$

Here the term $E(W^i)$ in equation 4 represents the error in the network output when a hidden node i is removed, as it is graphically shown in figure 4. The goal is to find a weight configuration that minimizes $E(W^i)$ for all nodes; minimization of the maximum $E(W^i)$ implies minimization of each of the $E(W^i)$. The performance constraints, as indicated in equation 5, capture the requirement that when all the nodes in the network are functional, for each input x^l to the network the output y^l must be equal the corresponding desired output d^l. The objective is to determine a weight matrix such that the network not only classifies the patterns or performs the computational task as desired, but it is also maximally fault tolerant.

The main difficulty with a *minimax* optimization problem is that the objective function is in general nondifferentiable; hence well-known gradient-based methods cannot be used to solve such problems. Fault tolerance posed as an optimization problem does not explicitly add, replicate, any spatial redundancy to the network, nor does it involves the modification of standard training algorithms.

Neti *et* al. [66] proposed the concept of maximally fault tolerant feedforward neural networks, where the determination of weights was formulated as a nonlinear optimization problem. Authors provided a first formalization of the concept of epsilon-fault tolerant neural network, as introduced in section III. Their method selects the weights that perform the required computation and have the additional property that whenever any single hidden unit is deleted, the faulty network continues to perform the computation satisfactorily. Pattern recognition examples were analyzed showing that uniformly fault tolerant solutions exist in a network with a single hidden layer. Uniformity of fault tolerance is a measure of the extent to which the computation performed is evenly distributed through neurons in the hidden layer. To maximize the number of different single hidden units while finding the weights that minimize the error, a successive quadratic programming algorithm to calculate the weights was used in this work.

Deodhare textit et al. [103] presented a technique for generating feedforward networks tolerant to the loss of a node and its associated weights. The problem was also formulated as a *minimax* optimization problem and two different solutions were addressed: i) optimization is converted to a sequence of unconstrained least-squares optimization problems, whose solutions converge to the solution of the original problem. Then a gradient-based minimization technique is applied to the unconstrained minimization, ii) the problem is converted to single unconstrained problem equivalent to the original one. The methods proposed here lead to networks that exhibit a partial degree of fault tolerance. However, authors claim that those methods might be extended to ensure complete fault tolerance. Both in terms of time and space, achieving fault tolerance in this approach is significantly more expensive than previous methods. Authors argue that such problem mainly arises due to the choice of using conventional optimization algorithms to perform the minimization, thus more advanced methods should be further explored.

Considering that genetic algorithms are a powerful tool for optimization, some works have employed them to search for a solution to fault tolerance. Zhou and Chen [104] employed a genetic algorithm to improve the tolerance of feedforward neural networks against an open fault, where a hidden node and its associated weights are considered to be faulty. The proposed method does not explicitly add any redundancy to the network as other works in this category, nor does it modify conventional training algorithm. The proposed method follows the key idea of genetic algorithms, maintain a population of neural networks, then use some fault-tolerant measures,

TABLE 5. Summary of some representative works that address active fault tolerance of neural networks.

Reference	Fault model	Neural model	Comment
[106]	Weight perturbation	Feedforward	Once a fault is detected, weight shifting is invoked. Fault detection is a black box
[107]	Weight perturbation	Self-organizing map	Weight shifting to recover when some faulty link/neurons appear
[108]	Stuck-at	Hierarchical cortical	Periodically retraining of the application is needed to adapt to the new configuration
[109]	Timing variations	Feedforward	If timing errors significantly affect the output results retraining is applied
[110]	Stuck-at	Spiking neurons	Biological-inspired self-repair mechanism based on astrocytes

fitness, to promote the population to evolve good fault tolerance. Experiments show that the proposed method improves fault tolerance as well as the generalization ability of neural networks. Similarly, the approaches proposed in [111] and [112] use genetic algorithms as the optimization method of choice for obtaining fault tolerant multilayer neural networks. However, in the first work, a fault tolerant multi-layer neural network, employing both hardware redundancy and weight retraining in order to realize self-recovering neural networks, is proposed, i.e., it provides active fault-tolerance.

B. ACTIVE FAULT TOLERANCE

In this section, we present a brief review of some representative works that propose methods/techniques to achieve active fault tolerance in neural models when targeted to physical hardware implementations. Such works are summarized in table 5. Under this approach, low-latency fault detection and recovery techniques are required to ensure that a neural network is reset into a fault-free and consistent state after a fault has occurred and propagated.

Khunasaraphan *et* al. [106] introduced a self-recovery mechanism called *weight shifting* applied to feedforward neural networks and outlined a hardware architecture for implementing this technique. Once a link or a neuron is detected to be faulty, weight shifting is invoked. If some links of a given neuron are faulty, their weights are shifted to other fault-free links of the same neuron. In case of a complete faulty neuron, all the output links of the output neuron are considered to be faulty. The fault detection circuit for links/neurons was considered as a black box in this work. In a further work, [107], authors applied weight shifting to recover a self-organized maps after some faulty link/neurons occur during operation without retraining or hardware repair. The fault detection or self-testing technique is not clearly described but it is rather suggested that information coding techniques can be employed for such purpose. Results were presented and validated for a 2×2 self-organizing map. To address fault detection, Tanprasert *et* al. [113] presented a technique for detecting the faulty links and determining the faulty weights in single-output two-layered feedforward neural networks by using a set of predefined probing vectors as inputs.

Hashmi *et* al. [108] described a biologically plausible computational model of cortical perceptual maps and highlighted its inherent fault tolerance. They demonstrated that such cortical maps can tolerate some failure modes that can occur in commodity GPGPUs systems. The model software implementation can intrinsically preserve its functionality in the presence of faulty hardware, considering a stuck-at fault model, without requiring any reprogramming or recompilation. Periodically retraining of the application is needed to adapt to the new configuration, but without explicitly specifying that configuration; the learning process will automatically adjust to the faulty hardware. Fault injection experiments validated that such systems are inherently far more tolerant to permanent faults than conventional ones and that can be applied for the robust implementation of tasks on future computing systems built of faulty components.

Deng *et* al. [109] studied the impacts of timing errors in hardware neural networks, feedforward multilayer models, to suit the de-facto distribution of timing variation in each individual chip. They proposed a timing variation-aware retraining method, thereby mitigating the negative effects of timing violations through the intrinsic resilience of neural networks. Once the accumulated delay of all gates and wires along a path exceeds the specified clock cycle, a timing violation occurs. Specifically, when timing errors significantly affect the output results, they retrain the neural accelerators to change their weights. In this way circumventing excessive timing errors. Experimental results show that timing errors in neural accelerators can be well tolerated for different applications.

Finally, Naeem *et* al. [110] demonstrated that a network model of spiking neurons can self-repair in the presence of a uniform and significant (up to 80%) fault distribution. Recall that, neurons of the central nervous system interact primarily with action potentials or spikes, which are stereotyped electrical impulses [120]. In this model, faults manifest themselves as silent or near silent neurons because of a sudden drop in probability of release (PR) at synaptic sites. The enhancement of PR, associated with non-faulty or healthy synapses, by the indirect retrograde signal is a key step in the repair process. Authors hypothesize that this repair strategy is effective for a nonuniform fault distribution because the proposed repair mechanism relies on the level of neural activity within the network being sufficient to maintain calcium oscillations across all astrocytes. Authors point out that by moving toward a more astrocentric computing paradigm that captures the self-repairing capability of the brain, it will open up an entirely

TABLE 6. Fault tolerance in non feedforward neural networks.

Reference	Neural network model	Comment
[67]	Recurrent networks	Aimed at solving optimization problems, optimization networks exhibit partial fault tolerance
[9], [114], [115]	Associative memories	The degree of fault tolerance in such models strongly depend on the assumed physical faults and the nature of the stored information
[116], [117]	Radial basis function	Established some boundaries for the degradation factor in the memory capacity, and margins on noise
[118], [119]	Self organizing maps	SOMs can eventually organize itself for fault tolerance if the defective neuron stuck- output is larger than a critical-stuck-output

new generation of brain-inspired autonomous computing systems.

C. BEYOND FEEDFORWARD NEURAL NETWORKS

As it was previously pointed out, most revised works have been focused in feedforward neural networks and few attempts have been made to analyze some other neural models for fault tolerance such as recurrent networks, RBF networks, associative memories, or self-organizing maps (SOM). This section provides a brief review of some of such works, summarized in table 6, even though these do not propose any technique to improve fault tolerance in those models.

Protzel *et* al. [67] investigated fault tolerance of continuous time recurrent neural networks aimed at solving optimization problems. Networks were subjected to up to 13 simultaneous stuck-at faults for sizes up to 900 neurons. Fault locations were randomly selected but no two stuck-at-1 faults are allowed to occur within the same row or column, as this automatically preclude a valid solution. In their study, mixed stuck-at faults were not considered so as to distinguish and compare the effect of a different fault type in the same locations. They defined a conditional performance measure by viewing the faults as constraints to the problem. According to results, optimization networks exhibit partial fault tolerance, which is achieved without the explicit use of redundant components.

Nijhuis and Spaanenburg [9] studied fault tolerance of neural associative memories using the Hopfield model under stuck-at-0 and stuck-at-1 faults in the neuron output, broken connections and weight deviation of their nominal values. The fault tolerance was considered to be the probability that the network will still function if there are x faulty neurons and y faulty connections. Results showed that the degree of fault tolerance in such models strongly depend on the assumed physical faults and the nature of the stored information: number of stored patterns, their correlation and desired radius of attraction. For small fault rates (below 20% of the total number of connections) such neural models proved to be less vulnerable to broken connections, whereas for large fault rates deviations in the connection strengths have less influence on the functioning of the network.

Leung *et* al. [114] studied the effect of multiplicative noise and open weight faults on the performance of bidirectional associative memories (BAM). Recall that a BAM is a two-layer heteroassociator that stores a prescribed set of bipolar pattern pairs, namely library pairs. They studied how many number of pattern pairs can be stored in a faulty BAM even when there are some errors in the initial stimulus patterns. They established some boundaries for the degradation factor in the memory capacity, and margins on noise providing a chance to recall the desired library pair. Leduc-Primeau *et* al. [115] studied fault-tolerant associative memories based on c-partite graphs. By analytical and simulation results they show that these associative memories can be made resilient to faults by modifying the retrieval algorithm. Faults were grouped into those affecting the representation of the graph's adjacency relationships, and those affecting the state of the retrieval algorithm. For a case study, the memory retains 88% of its efficiency when 1% of the storage cells are faulty, or 98% when 0.1% of the binary outputs of the retrieval algorithm are faulty.

Parra and Català [116] presented a sensitivity analysis for determining the most critical neural elements in a RBF network. Parametric faults in neural elements such as weights and biases, which involves a tolerance parameter related to multiplicative and additive noise in weights, were considered. The RMSE was used to calculate the approximation quality and as a measure of fault tolerance. The RBF networks and their topologies consisted in one hidden layer with Gaussian functions and one output layer with linear weighted addition. They concluded that the larger the weights, the worse the fault tolerance. Eickhoff and Rückert [117] studied the robustness of RBF networks in noisy and unreliable environments. If the network parameters are constrained, upper bounds on the MSE can be determined under noise contaminated parameters and inputs. A technique to identify sensitive neurons and to apply methods to increase reliability or to reduce noise to high sensitive neurons was only evaluated. Sum *et* al. [121] studied two different node-fault-injection-based on-line learning algorithms, including (1) injecting multinode fault during training and (2) weight decay with injecting multinode fault. By fault injection, either fault or noise is introduced to a network before each step of training. Proofs on the convergences of two node-fault-injection-based on-line training RBF methods have been shown and their corresponding objective functions were deduced. Some other fault tolerant learning methods for RBF have been proposed such as those reported in [94], [95], and [121]–[124].

Yasunaga *et* al. [118] studied the fault tolerant capability of SOM in the presence of defective neurons.

under stuck-at faults. It was shown that defective SOMs can eventually organize itself for fault tolerance if the defective neuron stuck-output is larger than a critical-stuck-output. Only a linear array was analyzed and discussed, where defective neurons, were concentrated in one place in the array, forming what is called a defective-neuron cluster. In the experiments, 100 neurons, including six defective ones, were used. Talumassawatdi and Lursinsap [119] addressed fault tolerance improvement in SOMs by using a technique of fault immunization of the synaptic connections. Stuck-at-*a* faults, where *a* is a real value, were considered. Only one neuron can be faulty at any time, but no restriction on the number of faulty links of the neuron is assumed. Weights are immunized by adding a constant value that is increased or decreased as much as possible without creating any misclassification. Fault immunization is formulated as an optimization problem on finding the corresponding constant value for each neuron.

V. OPEN CHALLENGES

Research directions that merit further exploration to enhance fault tolerance in neural computing need to pointed out. Some of these open issues - certainly not an exhaustive list - include the following:

1) *Novel fault models*, more realist models need to be developed based on a deep understanding of modern fabrication technologies. Moreover, due to the variety of failure modes and the high level of interplay involved, combinatorial strategies may be needed to obtain consistent, high quality fault tolerant neural mechanisms.
2) *Fault tolerance at architecture and application level*, efficient large scale fault tolerance mechanisms need to be designed while also leveraging the intrinsic characteristics of underlying neural models. Scalability of fault tolerance, currently is limited to small problem sizes, but neural networks are rapidly growing in size and complexity for emerging machine learning applications.
3) *Fault tolerance across different neural models*, fault tolerance enhancement techniques applicability is ad-hoc to specific neural model topologies, particularly for feedforward neural networks; ensuring different fault tolerance techniques operating at different levels is a major research challenge.
4) *Integration and coordination between different approaches* need be promoted. Even if a neural network may have its rather inherent fault tolerance, some additional mechanisms to enhance it should be incorporated for an specific implementation media.
5) *Interdisciplinary interaction*, reinforce potential interactions between neuroscience, computational neuroscience and neural networks so as to look for biological plausible fault tolerant mechanisms that allows to explore active fault tolerance principles and self repairing mechanisms.

VI. CONCLUSIONS

The connectionist and distributed nature of neural computing potentially leads to graceful degradation as exhibited by most neural networks models, i.e., neural networks will not suffer catastrophic failure, but any fault will influence the output to some degree since all components take part in the computational task. Considering neurons and synapses as physical entities that can fail independently is key in a truly distributed and scalable computing model with biological plausibility. Fault tolerance is not inherent within neural networks and is far from being complete; it does need to be specifically designed and built into the models. Approaching failure in neural network models is possible to be detected by using a continuous measure. The additional computational complexity that arises in fault tolerance enhancement techniques is absent in standard neural networks design.

In this paper, we presented a review of the main passive techniques used for improving the fault tolerance of neural networks, mainly for small size feedforward multilayer models, that exploit redundancy and fault masking. However, the obtained results for feedforward multilayer neural networks currently are both of relevance and a great opportunity for further exploration, since such networks are the quintessential elements of deep learning models, which have shown state-of-the-art performance on real-world artificial intelligence applications. The reviewed works have been categorized into three main categories based on key parameters and mechanisms to highlight their similarities and differences in pursuing fault tolerance in neural networks. The key role of fault tolerance in future computing systems has been highlighted by an important body of work. From a pragmatic point of view, the potential fault-tolerant property of neural models will be crucial to the success of attempts to integrate large scale neural models onto silicon, e.g. neural hardware accelerators, when problems of yield become unavoidable.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.
[2] C. Alippi, "Selecting accurate, robust, and minimal feedforward neural networks," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 49, no. 12, pp. 1799–1810, Dec. 2002.
[3] H. R. Mahdiani, S. M. Fakhraie, and C. Lucas, "Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1215–1228, Aug. 2012.
[4] S. Srinivasan and C. F. Stevens, "Robustness and fault tolerance make brains harder to study," *BMC Biol.*, vol. 9, p. 46, Jun. 2011.
[5] W. Maass, "Noise as a resource for computation and learning in networks of spiking neurons," *Proc. IEEE*, vol. 102, no. 5, pp. 860–880, May 2014.
[6] T. Sejnowksi and T. Delbruck, "The language of the brain," *Sci. Amer.*, vol. 307, pp. 54–59, Oct. 2012.
[7] W. Maass, "To spike or not to spike: That is the question," *Proc. IEEE*, vol. 103, no. 12, pp. 2219–2224, Dec. 2015.
[8] P. Chandra and Y. Singh, "Fault tolerance of feedforward artificial neural networks—A framework of study," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 1. Jul. 2003, pp. 489–494.
[9] J. A. G. Nijhuis and L. Spaanenburg, "Fault tolerance of neural associative memories," *IEE Proc. E-Comput. Digit. Techn.*, vol. 136, no. 5, pp. 389–394, Sep. 1989.

[10] S. S. Venkatesh, "The science of making ERORS: What error tolerance implies for capacity in neural networks," *IEEE Trans. Knowl. Data Eng.*, vol. 4, no. 2, pp. 135–144, Apr. 1992.
[11] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *IEEE Micro*, vol. 33, no. 3, pp. 16–27, May 2013.
[12] Z. Du, A. Lingamneni, Y. Chen, K. V. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of neural networks for designing highly energy efficient accelerators," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 8, pp. 1223–1235, Aug. 2015.
[13] G. Volanis, A. Antonopoulos, A. A. Hatzopoulos, and Y. Makris, "Toward silicon-based cognitive neuromorphic ICs—A survey," *IEEE Design Test*, vol. 33, no. 3, pp. 91–102, Jun. 2016.
[14] J. Arlat, Z. Kalbarczyk, and T. Nanya, "Nanocomputing: Small devices, large dependability challenges," *IEEE Security Privacy*, vol. 10, no. 1, pp. 69–72, Jan. 2012.
[15] Z. Wang, K. H. Lee, and N. Verma, "Overcoming computational errors in sensing platforms through embedded machine-learning kernels," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 8, pp. 1459–1470, Aug. 2015.
[16] D. Terry, "Toward a new approach to IoT fault tolerance," *Computer*, vol. 49, no. 8, pp. 80–83, Aug. 2016.
[17] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Autom. Stud.*, vol. 34, pp. 43–98, 1956.
[18] D. D. Thaker, R. Amirtharajah, F. Impens, I. L. Chuang, and F. T. Chong, "Recursive TMR: Scaling fault tolerance in the nanoscale era," *IEEE Design Test Comput.*, vol. 22, no. 4, pp. 298–305, Jul. 2005.
[19] K. Roy, B. Jung, D. Peroulis, and A. Raghunathan, "Integrated systems in the more-than-Moore era: Designing low-cost energy-efficient systems using heterogeneous components," *IEEE Design Test*, vol. 33, no. 3, pp. 56–65, Jun. 2016.
[20] M. Haselman and S. Hauck, "The future of integrated circuits: A survey of nanoelectronics," *Proc. IEEE*, vol. 98, no. 1, pp. 11–38, Jan. 2010.
[21] A. Eghbal, P. M. Yaghini, N. Bagherzadeh, and M. Khayambashi, "Analytical fault tolerance assessment and metrics for TSV-based 3D network-on-chip," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3591–3604, Dec. 2015.
[22] A. Rahimi, L. Benini, and R. K. Gupta, "Variability mitigation in nanometer CMOS integrated systems: A survey of techniques from circuits to software," *Proc. IEEE*, vol. 104, no. 7, pp. 1410–1448, Jul. 2016.
[23] S. Mittal and J. S. Vetter, "A survey of techniques for modeling and improving reliability of computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 1226–1238, Apr. 2016.
[24] A. DeHon, N. Carter, and H. Quinn. (Mar. 2011). "CCC cross-layer reliability visioning study," Nat. Sci. Found., USA, Tech. Rep. LA-UR 10-08387. [Online]. Available: http://www.relxlayer.org
[25] P. M. Furth and A. G. Andreou, "On fault probabilities and yield models for VLSI neural networks," *IEEE J. Solid-State Circuits*, vol. 32, no. 8, pp. 1284–1287, Aug. 1997.
[26] J. M. Shalf and R. Leland, "Computing beyond Moore's law," *Computer*, vol. 48, no. 12, pp. 14–23, Dec. 2015.
[27] A. E. Barbour and A. S. Wojcik, "A general constructive approach to fault-tolerant design using redundancy," *IEEE Trans. Comput.*, vol. 38, no. 1, pp. 15–29, Jan. 1989.
[28] M. Peercy and P. Banerjee, "Fault tolerant VLSI systems," *Proc. IEEE*, vol. 81, no. 5, pp. 745–758, May 1993.
[29] V. Piuri, "Analysis of fault tolerance in artificial neural networks," *J. Parallel Distrib. Comput.*, vol. 61, no. 1, pp. 18–48, 2001.
[30] A.-D. Almasi, S. Wozniak, V. Cristea, Y. Leblebici, and T. Engbersen, "Review of advances in neural networks: Neural design technology stack," *Neurocomputing*, vol. 174, pp. 31–41, Jan. 2016.
[31] N. C. Hammadi and H. Ito, "Improving the performance of feedforward neural networks by noise injection into hidden neurons," *J. Intell. Robot. Syst.*, vol. 21, no. 2, pp. 103–115, 1998.
[32] P. J. Edwards and A. F. Murray, "Toward optimally distributed computation," *Neural Comput.*, vol. 10, no. 4, pp. 987–1005, Sep. 1998.
[33] N. S. Merchawi, S. R. T. Kumara, and C. R. Das, "A probabilistic model for the fault tolerance of multilayer perceptrons," *IEEE Trans. Neural Netw.*, vol. 7, no. 1, pp. 201–205, Jan. 1996.
[34] X. Zeng and D. S. Yeung, "Sensitivity analysis of multilayer perceptron to input and weight perturbations," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1358–1366, Nov. 2001.
[35] M. Stevenson, R. Winter, and B. Widrow, "Sensitivity of feedforward neural networks to weight errors," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 71–80, Mar. 1990.
[36] Y. L. Cun, J. S. Denker, and S. A. Solla, "Advances in neural information processing systems," in *Optimal Brain Damage*, D. S. Touretzky, Ed. San Francisco, CA, USA: Morgan Kaufmann, 1990, pp. 598–605.
[37] J. L. Bernier, J. Ortega, E. Ros, I. Rojas, and A. Prieto, "A quantitative study of fault tolerance, noise immunity, and generalization ability of MLPs," *Neural Comput.*, vol. 12, no. 12, pp. 2941–2964, 2000.
[38] E. B. Tchernev, R. G. Mulvaney, and D. S. Phatak, "Investigating the fault tolerance of neural networks," *Neural Comput.*, vol. 17, no. 7, pp. 1646–1664, Jul. 2005.
[39] E. M. El Mhamdi and R. Guerraoui, "When neurons fail—Technical report," EPFL, Lausanne, Tech. Rep. EPFL-WORKING-217561, 2016.
[40] Y. Wang and A. Avižienis, "A unified reliability model for fault-tolerant computers," *IEEE Trans. Comput.*, vol. C-29, no. 11, pp. 1002–1011, Nov. 1980.
[41] A. Kulakov, M. Zwolinski, and J. S. Reeve, "Fault tolerance in distributed neural computing," *CoRR*, vol. abs/1509.09199, pp. 1–9, Sep. 2015.
[42] A. Avižienis, "Framework for a taxonomy of fault-tolerance attributes in computer systems," *SIGARCH Comput. Archit. News*, vol. 11, no. 3, pp. 16–21, Jun. 1983.
[43] V. P. Nelson, "Fault-tolerant computing: Fundamental concepts," *Computer*, vol. 23, no. 7, pp. 19–25, Jul. 1990.
[44] A. K. Somani and N. H. Vaidya, "Understanding fault tolerance and reliability," *Computer*, vol. 30, no. 4, pp. 45–50, Apr. 1997.
[45] P. G. Depledge, "Fault-tolerant computer systems," *IEE Proc. A-Phys. Sci., Meas. Instrum., Manage. Edu.-Rev.*, vol. 128, no. 4, pp. 257–272, May 1981.
[46] G. Buja and R. Menis, "Dependability and functional safety: Applications in industrial electronics systems," *IEEE Ind. Electron. Mag.*, vol. 6, no. 3, pp. 4–12, Sep. 2012.
[47] S. Gai, M. Mezzalama, and P. Prinetto, "A review of fault models for LSI/VLSI devices," *Softw. Microsyst.*, vol. 2, no. 2, pp. 44–53, Apr. 1983.
[48] J. Sosnowski, "Transient fault tolerance in digital systems," *IEEE Micro*, vol. 14, no. 1, pp. 24–35, Feb. 1994.
[49] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 389–402, Mar. 2009.
[50] N. Aymerich, S. D. Cotofana, and A. Rubio, "Adaptive fault-tolerant architecture for unreliable technologies with heterogeneous variability," *IEEE Trans. Nanotechnol.*, vol. 11, no. 4, pp. 818–829, Jul. 2012.
[51] K. C. Y. Mei, "Bridging and stuck-at faults," *IEEE Trans. Comput.*, vol. C-23, no. 7, pp. 720–727, Jul. 1974.
[52] G. Bolt, "Fault models for artificial neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 2. Nov. 1991, pp. 1373–1378.
[53] A. Pancholy, J. Rajski, and L. J. McNaughton, "Empirical failure analysis and validation of fault models in CMOS VLSI circuits," *IEEE Design Test Comput.*, vol. 9, no. 1, pp. 72–83, Mar. 1992.
[54] P. Gil, J. Arlat, H. Madeira, Y. Crouzet, T. Jarboui, and K. Kanoun, "Fault representativeness," Eur. Community Dependability Benchmarking Project, France, Tech. Rep. IST-200025425, 2002.
[55] D. de Andres, J. C. Ruiz, D. Gil, and P. Gil, "Fault emulation for dependability evaluation of VLSI systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 4, pp. 422–431, Apr. 2008.
[56] J. A. Abraham and W. K. Fuchs, "Fault and error models for VLSI," *Proc. IEEE*, vol. 74, no. 5, pp. 639–654, May 1986.
[57] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 3, pp. 583–602, Jun. 2003.
[58] S. Ghosh and K. Roy, "Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era," *Proc. IEEE*, vol. 98, no. 10, pp. 1718–1751, Oct. 2010.
[59] M. Y. C. Kao, K.-T. Tsai, and S.-C. Chang, "A fault detection and tolerance architecture for post-silicon skew tuning," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 7, pp. 1210–1220, Jul. 2015.
[60] M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein, "A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability," *IEEE Commun. Surveys Tuts.*, vol. 11, no. 2, pp. 106–124, 2nd Quart., 2009.
[61] R. Frei, R. McWilliam, B. Derrick, A. Purvis, A. Tiwari, and G. Di Marzo Serugendo, "Self-healing and self-repairing technologies," *Int. J. Adv. Manuf. Technol.*, vol. 69, no. 5, pp. 1033–1061, 2013.

[62] M. A. Breuer, "Multi-media applications and imprecise computation," in *Proc. 8th Euromicro Conf. Digit. Syst. Des. (DSD)*, Aug. 2005, pp. 2–7.
[63] M. Stanisavljević, A. Schmid, and Y. Leblebici, *Fault-Tolerant Architectures and Approaches*. New York, NY, USA: Springer, 2011, pp. 35–47.
[64] E. Dubrova, *Hardware Redundancy*. New York, NY, USA: Springer, 2013, pp. 55–86.
[65] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 446–456, Mar. 1995.
[66] C. Neti, M. H. Schneider, and E. D. Young, "Maximally fault tolerant neural networks," *IEEE Trans. Neural Netw.*, vol. 3, no. 1, pp. 14–23, Jan. 1992.
[67] P. W. Protzel, D. L. Palumbo, and M. K. Arras, "Performance and fault-tolerance of neural networks for optimization," *IEEE Trans. Neural Netw.*, vol. 4, no. 4, pp. 600–614, Jul. 1993.
[68] C. H. Sequin and R. D. Clay, "Fault tolerance in artificial neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 1. Jun. 1990, pp. 703–708.
[69] K. Mehrotra, C. K. Mohan, and S. Ranka, "Fault tolerance in neural networks," School Comput. Inf. Sci., Syracuse Univ., Syracuse, NY, USA, Tech. Rep. RL-TR-94-93, Jul. 1994.
[70] Y. Tohma and Y. Koyanagi, "Fault-tolerant design of neural networks for solving optimization problems," *IEEE Trans. Comput.*, vol. 45, no. 12, pp. 1450–1455, Dec. 1996.
[71] D. B. I. Feltham and W. Maly, "Physically realistic fault models for analog CMOS neural networks," *IEEE J. Solid-State Circuits*, vol. 26, no. 9, pp. 1223–1229, Sep. 1991.
[72] A. S. Orgenci, G. Dundar, and S. Balkur, "Fault-tolerant training of neural networks in the presence of MOS transistor mismatches," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 3, pp. 272–281, Mar. 2001.
[73] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," *SIGARCH Comput. Archit. News*, vol. 40, no. 3, pp. 356–367, Jun. 2012.
[74] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, Apr. 1997.
[75] J. L. Bernier, J. Ortega, E. Ros, I. Rojas, and A. Prieto, "Obtaining fault tolerant multilayer perceptrons using an explicit regularization," *Neural Process. Lett.*, vol. 12, no. 12, pp. 107–113, 2001.
[76] O. Fontenla-Romero, E. Castillo, A. Alonso-Betanzos, and B. Guijarro-Berdias, "A measure of fault tolerance for functional networks," *Neurocomputing*, vol. 62, pp. 327–347, Dec. 2004.
[77] R. Xu and D. Wunsch, II, "Survey of clustering algorithms," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 645–678, May 2005.
[78] M. N. Shirazi, M. N. Shirazi, and S. Maekawa, "The capacity of associative memories with malfunctioning neurons," *IEEE Trans. Neural Netw.*, vol. 4, no. 4, pp. 628–635, Jul. 1993.
[79] N. C. Hammadi and H. Ito, "On the activation function and fault tolerance in feedforward neural networks," *IEICE Trans. Inf. Syst.*, vols. E81-D, no. 1, pp. 66–72, 1998.
[80] L. C. Chu and B. W. Wah, "Fault tolerant neural networks with hybrid redundancy," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 2. Jun. 1990, pp. 639–649.
[81] M. D. Emmerson and R. I. Damper, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 788–793, Sep. 1993.
[82] 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. Apr. 1993, pp. 783–788.
[83] C.-T. Chin, K. Mehrotra, C. K. Mohan, and S. Rankat, "Training techniques to obtain fault-tolerant neural networks," in *Proc. 24th Int. Symp. Fault-Tolerant Comput. (FTCS) Dig. Papers*, Jun. 1994, pp. 360–369.
[84] F. M. Dias and A. Antunes, "Fault tolerance improvement through architecture change," in *Artificial Neural Networks*. Berlin, Germany: Springer, 2008, pp. 248–257.
[85] F. M. Dias, R. Borralho, P. Fontes, and A. Antunes, "FTSET–a software tool for fault tolerance evaluation and improvement," *Neural Comput. Appl.*, vol. 19, no. 5, pp. 701–712, 2010.
[86] P. Chandra and Y. Singh, "Feedforward sigmoidal networks—Equicontinuity and fault-tolerance properties," *IEEE Trans. Neural Netw.*, vol. 15, no. 6, pp. 1350–1366, Nov. 2004.
[87] 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.
[88] N. Wei, S. Yang, and S. Tong, "A modified learning algorithm for improving the fault tolerance of bp networks," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 1. Jun. 1996, pp. 247–252.
[89] P. J. Edwards and A. F. Murray, "Penalty terms for fault tolerance," in *Proc. Int. Conf. Neural Netw.*, vol. 2. Jun. 1997, pp. 943–947.
[90] N. C. Hammadi, T. Ohmameuda, E. Kaneko, and H. Ito, "Fault tolerant constructive algorithm for feedforward neural networks," in *Proc. Pacific Rim Int. Symp. Fault-Tolerant Syst.*, Dec. 1997, pp. 215–220.
[91] S. Cavalieri and O. Mirabella, "A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks," *Neural Netw.*, vol. 12, no. 1, pp. 91–106, Jan. 1999.
[92] Z.-H. Zhou, S.-F. Chen, and Z.-Q. Chen, "Improving tolerance of neural networks against multi-node open fault," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 3. 2001, pp. 1687–1692.
[93] D. Simon, "Distributed fault tolerance in optimal interpolative nets," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1348–1357, Nov. 2001.
[94] Y. Xiao, R.-B. Feng, C. S. Leung, and J. Sum, "Objective function and learning algorithm for the general node fault situation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 4, pp. 863–874, Apr. 2016.
[95] C.-S. Leung, W. Y. Wan, and R. Feng, "A regularizer approach for RBF networks under the concurrent weight failure situation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 6, pp. 1360–1372, Jun. 2017.
[96] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE*, vol. 78, no. 9, pp. 1481–1497, Sep. 1990.
[97] R. Reed, R. J. Marks, II, and S. Oh, "Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 529–538, May 1995.
[98] M. M. Islam, M. A. Sattar, M. F. Amin, X. Yao, and K. Murase, "A new adaptive merging and growing algorithm for designing artificial neural networks," *IEEE Trans. Syst., Man, B (Cybern.)*, vol. 39, no. 3, pp. 705–722, Jun. 2009.
[99] N. Hammadi and H. Ito, "A learning algorithm for fault tolerant feedforward neural networks," *IEICE Trans. Inf. Syst.*, vol. E80-D, no. 1, pp. 21–27, 1997.
[100] J. Bernier, J. Ortega, I. Rojas, and A. Prieto, "Improving the tolerance of multilayer perceptrons by minimizing the statistical sensitivity to weight deviations," *Neurocomputing*, vol. 31, nos.1–4, pp. 87–103, 2000.
[101] S.-K. Sin and R. J. P. DeFigueiredo, "Efficient learning procedures for optimal interpolative nets," *Neural Netw.*, vol. 6, no. 1, pp. 99–113, 1993.
[102] R. J. P. de Figueiredo, "An optimal matching-score net for pattern classification," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 3. Jun. 1990, pp. 909–916.
[103] D. Deodhare, M. Vidyasagar, and S. S. Keethi, "Synthesis of fault-tolerant feedforward neural networks using minimax optimization," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, pp. 891–900, Sep. 1998.
[104] Z.-H. Zhou and S.-F. Chen, "Evolving fault-tolerant neural networks," *Neural Comput. Appl.*, vol. 11, nos. 3–4, pp. 156–160, Jun. 2003.
[105] S. Bettola and V. Piuri, "High performance fault-tolerant digital neural networks," *IEEE Trans. Comput.*, vol. 47, no. 3, pp. 357–363, Mar. 1998.
[106] C. Khunasaraphan, K. Vanapipat, and C. Lursinsap, "Weight shifting techniques for self-recovery neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 4, pp. 651–658, Jul. 1994.
[107] C. Khunasaraphan, T. Tanprasert, and C. Lursinsap, "Recovering faulty self-organizing neural networks: By weight shifting technique," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 3. Jun. 1994, pp. 1513–1518.
[108] A. Hashmi, H. Berry, O. Temam, and M. Lipasti, "Automatic abstraction and fault tolerance in cortical microachitectures," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2011, pp. 1–10.
[109] J. Deng *et al.*, "Retraining-based timing error mitigation for hardware neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2015, pp. 593–596.
[110] M. Naeem, L. J. McDaid, J. Harkin, J. J. Wade, and J. Marsland, "On the role of astroglial syncytia in self-repairing spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 10, pp. 2370–2380, Oct. 2015.
[111] E. Sugawara, M. Fukushi, and S. Horiguchi, "Fault tolerant multi-layer neural networks with ga training," in *Proc. 18th IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, Nov. 2003, pp. 328–335.
[112] F. Su, P. Yuan, Y. Wang, and C. Zhang, "The superior fault tolerance of artificial neural network training with a fault/noise injection-based genetic algorithm," *Protein Cell*, vol. 7, no. 10, pp. 735–748, 10 2016.
[113] T. Tanprasert, C. Tanprasert, and C. Lursinsap, "Probing technique for neural net fault detection," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 2. Jun. 1996, pp. 1001–1005.

[114] A. C.-S. Leung, P. F. Sum, and K. Ho, "The effect of weight fault on associative networks," *Neural Comput. Appl.*, vol. 20, no. 1, pp. 113–121, 2011.
[115] F. Leduc-Primeau, V. Gripon, M. G. Rabbat, and W. J. Gross, "Fault-tolerant associative memories based on c-partite graphs," *IEEE Trans. Signal Process.*, vol. 64, no. 4, pp. 829–841, Feb. 2016.
[116] X. Parra and A. Català, "Sensitivity analysis of radial basis function networks for fault tolerance purposes," in *Foundations and Tools for Neural Modeling*. Alicante, Spain: Springer, 1999, pp. 566–572.
[117] R. Eickhoff and U. Rückert, "Robustness of radial basis functions," *Neurocomputing*, vol. 70, nos. 16–18, pp. 2758–2767, 2007.
[118] M. Yasunaga, I. Hachiya, K. Moki, and J. H. Kim, "Fault-tolerant self-organizing map implemented by wafer-scale integration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 2, pp. 257–265, Jun. 1998.
[119] R. Talumassawatdi and C. Lursinsap, "Fault immunization concept for self-organizing mapping neural networks," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 9, no. 6, pp. 781–790, 2001.
[120] R. Brette, "Philosophy of the spike: Rate-based vs. spike-based theories of the brain," *Frontiers Syst. Neurosci.*, vol. 9, p. 151, Nov. 2015.
[121] J. Sum, C.-S. Leung, and K. Ho, *On Node-Fault-Injection Training of an RBF Network*. Berlin, Germany: Springer, 2009, pp. 324–331.
[122] J. Pajarinen, J. Peltonen, and M. A. Uusitalo, "Fault tolerant machine learning for nanoscale cognitive radio," *Neurocomputing*, vol. 74, no. 5, pp. 753–764, 2011.
[123] K. Ho, C.-S. Leung, and J. Sum, "Training RBF network to tolerate single node fault," *Neurocomputing*, vol. 74, no. 6, pp. 1046–1052, 2011.
[124] R. Martolia, A. Jain, and L. Singla, "Analysis & survey on fault tolerance in radial basis function networks," in *Proc. Int. Conf. Comput., Commun. Autom. (ICCCA)*, May 2015, pp. 469–473.
[125] R.-B. Feng, Z.-F. Han, W. Y. Wan, and C.-S. Leung, "Properties and learning algorithms for faulty RBF networks with coexistence of weight and node failures," *Neurocomputing*, vol. 224, pp. 166–176, Feb. 2017.

CESAR TORRES-HUITZIL received the master's degree in electronics and the Ph.D. degree in computer science from the National Institute for Astrophysics Optics and Electronics in 2003 and 1998, respectively. He is currently a Researcher with the Information Technology Laboratory, Center for Research and Advanced Studies in the research unit located in Tamaulipas, Mexico. His main research lines are around reconfigurable computing and the computational applications of FPGA devices in different domains such as computer vision, digital signal processing, and neural computing.

BERNARD GIRAU received the Ph.D. degree in computer science from the Ecole Normale Superieure de Lyon in 1999. He is currently a Full Professor of computer science with Université de Lorraine and a Researcher with the Biscuit Team, LORIA Lorraine Laboratory, Nancy, France. His current research interest includes embedded parallel connectionism and bio-inspired neural models for visual perception.