Liu, Shanshan; Reviriego, Pedro; Lombardi, Fabrizio (2022). Selective Neuron Re-Computation (SNRC) for Error-Tolerant Neural Networks. *IEEE Transactions on Computers*, 71(3), pp.: 684-695.

DOI: https://doi.org/10.1109/TC.2021.3056992

Selective Neuron Re-Computation (SNRC) for Error-Tolerant Neural Networks

Shanshan Liu, *Member, IEEE*, Pedro Reviriego, *Senior Member, IEEE* and Fabrizio Lombardi, *Fellow, IEEE*

***Abstract*—Artificial Neural networks (ANNs) are widely used to solve classification problems for many machine learning applications. When errors occur in the computational units of an ANN implementation due to for example radiation effects, the result of an arithmetic operation can be changed, and therefore, the predicted classification class may be erroneously affected. This is not acceptable when ANNs are used in many safety-critical applications, because the incorrect classification may result in a system failure. Existing error-tolerant techniques usually rely on physically replicating parts of the ANN implementation or incurring in a significant computation overhead. Therefore, efficient protection schemes are needed for ANNs that are run on a processor and used in resource-limited platforms. A technique referred to as Selective Neuron Re-Computation (SNRC), is proposed in this paper. As per the ANN structure and algorithmic properties, SNRC can identify the cases in which the errors have no impact on the outcome; therefore, errors only need to be handled by re-computation when the classification result is detected as unreliable. Compared with existing temporal redundancy-based protection schemes, SNRC saves more than 60% of the re-computation (more than 90% in many cases) overhead to achieve complete error protection as assessed over a wide range of datasets. Different activation functions are also evaluated.**

***Index Terms*—Neural networks, machine learning, sigmoid, error-tolerance.**

I. Introduction

Artificial Neural Networks (ANNs) are used in machine learning to perform supervised and unsupervised learning. Based on the structure that models the neural networks in the human brain, ANNs simulate the brain to perform particular tasks; they have been applied in a wide range of areas, such as image classification of vehicles and medical diagnosis or anomaly detection in networks [1].

ANNs are typically arranged as a number of layers of neurons that connect the inputs (or features) to the outputs. Initially, only two layers of neurons (one for the inputs and another for the outputs) have been used. This however limits the ability of the ANN to discriminate data that is linearly separable in the feature space. To deal with more complex problems, additional layers of neurons are usually needed; these layers are commonly referred to as hidden layers. For example, a large number of layers are used in Convolutional Neural Networks (CNNs) that are widely used for image classification [2], [3]. The Multi-Layer Perceptron (MLP) that uses just a single hidden layer can also provide good results for many simpler classification problems, while achieving in most cases, a higher accuracy than other classifiers, such as the *k* Nearest Neighbors (*k*NNs) or Support Vector Machines (SVMs) [4]. Since the number of layers affects the computational complexity of the ANNs and greatly influences the platform used for implementation, MLP is very attractive for resource-constrained systems, such devices for the Internet of Things (IoT).

ANNs with a small number of neurons (e.g., MLP) can be implemented in software running on a processor and achieving a reasonable speed. However, ANNs that have many hidden layers (e.g., CNN) and thus neurons, are computationally expensive, so usually taking a long time for computation when implemented in software. In recent years, acceleration of ANNs using FPGAs or even dedicated ASICs has been widely investigated [5]-[7]. Accelerators are needed in systems in which complex ANNs are used to perform many classifications per second. However, for simpler ANNs or when the number of classifications to be performed is small, software implementations are attractive, because they provide additional flexibility and do not incur in additional hardware for acceleration [8].

Reliable operation is an important aspect of ANNs, especially when they are used in safety-critical applications [9], [10]. Regardless of the platform used for implementation, an ANN can be affected by several errors or failures [11]. For example, when a radiation particle hits an electronic device, it can cause several effects, ranging from permanent damage to soft errors. Radiation can come from the environment, but also from the material used to manufacture the integrated circuit. Environmental radiation is a major issue for devices that operate in harsh environments like space, but it also causes soft errors on terrestrial systems. So, when a particle strikes the device, extra charge is generated; this may modify the voltage of a node, upsetting a value stored in the storage units (i.e., the single bit upset effect), or introducing an unexpected current

Manuscript received October 6, 2020, revised December 23, 2020 and January 27, 2021, and accepted January 31, 2021. This research was supported by NSF grants CCF-1953961 and 1812467, by the ACHILLES project PID2019-104207RB-I00 and the Go2Edge network RED2018-102585-T funded by the Spanish Ministry of Science and Innovation and by the Madrid Community research project TAPIR-CM P2018/TCS-4496.

S. Liu and F. Lombardi are with Northeastern University, Dept. of ECE, Boston, MA 02115, USA (email: ssliu@coe.neu.edu, lombardi@ece.neu.edu).

P. Reviriego is with Universidad Carlos III de Madrid, Av. De la Universidad 30, 28911 Leganés, Madrid, Spain (email: revirieg@it.uc3m.es).

pulse in combinational circuits (i.e., the single bit transient effect). These errors can lead to data corruption and eventually system failure. If errors occur in the inference process of an ANN, they can change the value of a neuron, or the stored parameters in the memories of the ANN [12]; therefore, errors may have an impact on the final result (e.g., the predicted class when an ANN is used for classification). To ensure that the ANN continues to operate correctly in the presence of errors, an error-tolerant design is therefore needed.

Error/fault tolerance of ANNs has been widely studied; critical connections (so, with larger weights) have been shown to be more prone to lead to an incorrect output in the event of an error, thus modifications for the larger weights have been extensively treated in the technical literature. For example, in [13], [14] the connections with large weights are replaced by duplicated connections, but each with half of the weight to reduce the impact of an error. Other schemes try to reduce the impact of connections with large weights during the training phase, because often they significantly affect the classification result [15], [16]. Another approach to achieve error tolerance [17] has introduced errors during the training phase, such that the ANN has been trained to correct them. Selective hardening of critical hardware in ANN implementations (such as latches) has also been proposed [9]. However, these protection solutions usually need to change either the algorithm, or the implementation of the ANN, so not desirable when the ANN implementation is fixed and used for inference. An alternative that does not need to change the algorithm is based on temporal redundancy (TR); it computes the classification result of the ANN twice to detect errors and a third time when needed to correct them [18]. However, this scheme always incurs in a significant computation overhead, because it needs at least to double the number of operations. A fault management scheme of [19] can reduce the re-execution overheads, but it is applied to CNNs used in some applications that can tolerate a specific degree of inexactness (e.g., image processing). Another technique [9] (implemented in software) detects potential errors by checking if the output of the neurons exceeds the maximum value observed during error-free operation. This incurs in a low protection overhead when applied to deep ANNs, but it cannot detect all errors and thus, it only provides partial protection in shallow NNs (this aspect will be discussed and assessed in a latter section of this paper).

In this paper, the protection of ANNs implemented in software (so running on a processor) is studied; in particular, the improvement of a TR-based scheme as baseline is considered. The analysis of the ANN and its algorithmic properties is utilized to propose Selective Neuron Re-Computation (SNRC) as an efficient scheme that can reduce the re-computation overhead for error detection, while ensuring a reliable result. The scheme has been evaluated on several publicly available datasets; the results show that for most datasets, a significant reduction in the re-computation effort is achieved.

The rest of this paper is organized as follows. Section II introduces Artificial Neural Networks (ANNs); computational and storage errors in the implementation and commonly used error-tolerant techniques are also reviewed. In Section III, the error model considered in this paper is discussed; then the proposed technique, namely Selective Neuron Re-Computation (SNRC), is discussed in Section IV to efficiently handle computational errors in the inference process of ANNs. The effectiveness of the proposed technique is then evaluated and compared with traditional solutions in Section V using simulation; the extension of SNRC for ANNs with different activation functions is discussed in Section VI. Finally, the paper ends in Section VII with the conclusion.

II. Preliminaries

This section first provides a brief description of ANNs; then errors in the computation process for neurons in each layer and their impact on the classification result are discussed.

A. Artificial Neural Networks (ANNs)

The human brain is composed of neurons that are connected through synapses for transferring signals between them to form so-called neural networks. Neurons are also modeled in ANNs as the basic computation unit. The initial ANNs (i.e., the so-called perceptron) were proposed in 1958; they consist of two connected layers of neurons: the input and output layers [20]. In a perceptron, neurons in the input layer only transfer information, while neurons in the output layer perform computation on the information received from the input layer by considering different importance features (i.e., weights) of the information transferred from different neurons. The perceptron is also referred to as a single neural network and the weights can be trained to achieve good performance in the learning process under a simple linear classification. However, many types of empirical data are not linearly separable in the feature space, then single neural networks have been extended by adding a hidden layer between the input and output layers; in this scheme, the inputs can be analyzed in a higher dimensional space and complex non-linear issues can then be solved. This improved network is known as the Multi-Layer Perceptron (MLP) [21]; by utilizing a learning algorithm, such as Backward Propagation (BP), the computational complexity incurred by the extra hidden layer, is significantly reduced, so making MLP widely used to perform non-linear classification tasks. Although different neural networks with more hidden layers have been proposed (e.g., CNNs) to improve performance, MLP still remains attractive for many applications when taking into account both complexity and implementation issues.

Figure 1 shows an MLP that has the input layer, one hidden layer, and the output layer; the neurons in the input (hidden)

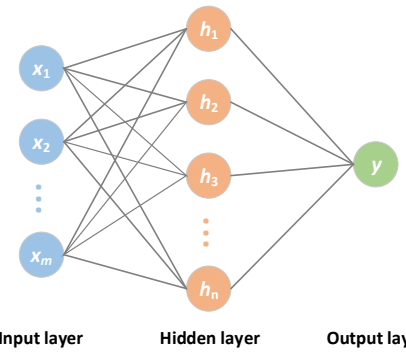

Figure 1 An MLP with one hidden layer.

layer are denoted as $\{x_i\}$ ($\{h_j\}$), where $i = 1, 2, \ldots, m$ ($j = 1, 2, \ldots, n$), and y is the neuron in the output layer. w_{ij}^1 is the weight of neuron x_i in the first layer (i.e., the input layer) to neuron h_j in the second layer (i.e., the hidden layer), and w_j^2 is the weight of neuron h_j in the hidden layer to the output; these weights are used to connect the neurons in neighboring layers. A bias unit (always storing a "1") exists in the input and hidden layers and its weight is given by b_l, where l is 1 (2) for the bias in the input (hidden) layer.

In general, the size of the input layer is equal to the valid feature dimension of the dataset, and the inputs are the normalized features. For each neuron in the hidden and the output layers, its value is calculated in two steps: i) a temporary value is calculated based on the neurons in the previous layer and the corresponding weights and bias; ii) the value is then compressed into a so-called specific scope by using a non-linear activation function Φ. There are several activation functions for handling different types of data and layers, such as the *sigmoid* function, the *tanh* function, the *rectifier (ReLU)* function and the *softmax* function. For binary classification, the *sigmoid* function can be utilized for both the hidden and the output layers, mapping any positive value or zero to the result in [0.5, 1) and any negative value to (0, 0.5). The *sigmoid* function has been shown to be well suited for MLP [22]. The computation process for a neuron in the hidden layer when using *sigmoid* as activation function is illustrated in Figure 2; computation in steps i) and ii) is given by Eqs. (1) and (2), respectively. The computation for the neuron in the output layer is similar; so, by feeding the neurons in the hidden layer, a *sigmoid* mapping is performed on the output neuron to obtain the final result (given in Eq. (3)). Since the *sigmoid* function always generates a value from 0 to 1, the output neuron with a value that is smaller than 0.5 refers to a class; when it is larger than or equal to 0.5, it refers to another class when the NN is used to compute a binary classification.

$$z_j(x) = \sum_{i=1}^{m} w_{ij}^1 \cdot x_i + b_1 \quad (1)$$

$$h_j = \Phi(z_j) = sigmoid(z_j) = \frac{1}{1+e^{-z_j}} \quad (2)$$

$$y = \Phi(h_j) = \frac{1}{1+e^{-\sum_{j=1}^{n} w_j^2 \cdot h_j - b_2}} \quad (3)$$

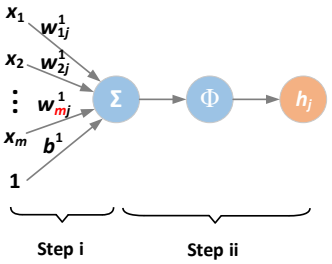

Figure 2 Computation for a neuron in the hidden layer.

B. Computational Errors and Protection

In a computing system, data processing and storage are prone to suffer different types of errors, such as for example radiation-induced soft errors [23], [24]. Errors in the computational units (referred to as computational errors in this paper) can modify the result of an operation (such as an arithmetic calculation), while errors in the memory elements (referred to as storage errors in this paper) can flip the stored data, causing data corruption; both these types of errors may lead to a system failure. Therefore, error-tolerant techniques are required to guarantee the integrity of a computing system, especially in many safety-critical applications.

Memories can be efficiently protected by utilizing error correction codes (ECCs) against storage errors [25]-[27]; through computing several ECC parity bits (as per the ECC encoding algorithm) and storing them with the original data, errors on each memory word can be detected or corrected as per the ECC decoding algorithm. Many memory/System-on-Chip products incorporate an appropriate ECC into the chip to provide protection against some storage errors [28], [29]; for example, Hamming codes, which can correct a single bit error at a low cost, are commonly used.

Redundancy is usually introduced to protect the computational units; this can be implemented spatially or temporally [30], [31]. In Spatial Redundancy (SR) based schemes, the circuit being protected is duplicated and the two outcomes are compared to detect errors in one of the replicated circuits. If needed, the original circuit is triplicated, then a majority voting among the three results is performed to guarantee a correct result. However, the hardware overhead incurred by SR-based techniques is rather large; it is more than 100% for error detection and at least 200% for error correction (due to the replication and the comparison logic). An alternative solution is Reduced Precision Redundancy (RPR) [32]. RPR uses multiple replicated copies with reduced precision (by truncating some least significant bits) and only one circuit with full precision; hence, the overhead can be reduced by sacrificing protection against some errors on the truncated bits. However, SR-based protection approaches cannot be implemented when the program is run on a processor or microcontroller, in which the hardware cannot be modified. In this case, temporal redundancy is generally utilized. Since computational errors are mostly transient and non-frequent, they are unlike to occur again when the program is run for a second time. Therefore, errors can be detected by re-computation of the program and then comparing the two output results; the computation is error-free if the two results fully match, otherwise an error is detected, and a third re-computation must be performed to obtain the correct result through majority voting. Similar to the SR-based approaches, the additional computational requirements (e.g., total execution time and power consumption) are rather severe for error detection/correction; these large overheads are unlikely to be viable when Machine Learning (ML) algorithms are used in resource-limited platforms, such in IoT devices.

Recently, algorithm-based error tolerance (ABET) has been proposed to protect some ML classifiers against computational errors; these techniques exploit the intrinsic redundancy of ML algorithms to achieve error-tolerance, thus significantly reducing the overall protection overhead. For example, the so-called Voting-Margin (VM) scheme (presented in [33]) protects *k*NNs against computational errors in the distance computation. As the classification result is reliable when voting among the set of *k*NNs has a margin, errors need to be detected

only when there is no VM. This scheme has also been investigated for *k*NNs and other ensemble classifiers, in particular Random Forests (RFs), when used for multiple classification tasks [34]; the work of [34] has shown that RFs are intrinsically robust under errors because in such classifiers, each single voter (e.g., a decision tree) performs the first round classification and then the results of multiple voters are used to perform a second round classification, making most errors unlikely to change the final result. However, RFs typically incur in a significant implementation overhead due to the large number of trees. Another ABET scheme known as Result Based Re-computation (RBR), has been proposed in [35] to protect the Support Vector Machine (SVM) against computational errors in the kernel function. RBR determines the conditions by which the kernel function for a given support vector must be re-computed based on the magnitude of the error; therefore, only the errors that introduce a deviation in the final result and then modify the predicted class, must be detected and corrected by re-computation. A principle similar to RBR is then applied in this paper to propose a protection scheme for ANNs; it only needs to perform re-computation for some neurons, while ensuring that the classification result is correct. This will be discussed in the following sections.

III. Error Model

When using the *sigmoid* function in ANNs for binary classification, the proposed error-tolerant scheme is based on the observation that when an error occurs, if neuron y in the output layer has a value that is in the same range (0, 0.5) or [0.5, 1) as the original error-free value, then the error has no impact on the classification result, i.e.:

- If $y < 0.5$ in the error-free case, errors causing $y\_{err} < 0.5$ (where y_{err} is the incorrect output neuron) cannot change the predicted class for the input data;
- If $y \geq 0.5$ in the error-free case, the classification result is always reliable for errors causing $y\_{err} \geq 0.5$.

The computational errors in the implementation of the considered network in this paper are classified in the following two types:

- **Type 1** errors that occur during the computation of a neuron in the hidden layer;
- **Type 2** errors that occur during the computation of a neuron in the output layer.

Each type of error can be caused by the incorrect computation in three possible operations; a Type 1 error is illustrated in Figure 3 as an example. In Figure 3, error a) occurs in the multiplication by the weight, error b) occurs in the sum of the inputs to the neuron, and error c) occurs during the *sigmoid* mapping.

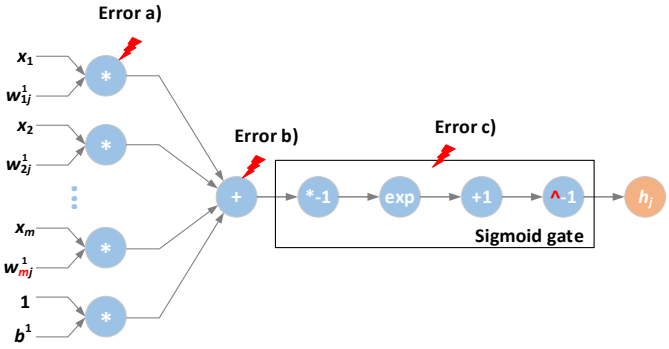

Figure 3 Type 1 errors in the computation for a neuron in the hidden layer.

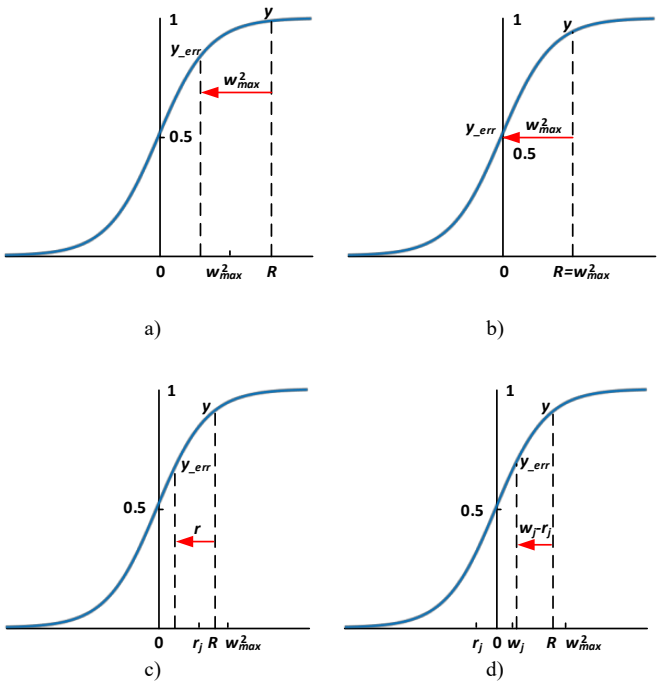

Figure 4 Impact of Type 1 errors that occur during the computation by a neuron in the hidden layer in different cases: a) $|R| \geq |w_{max}|$; b) $|R| = |w_{max}|$; c) $|w_{max}| \geq |R| \geq |r_j|$ and $R \cdot r_j \geq 0$; d) $|w_{max}| \geq |R| \geq |w_j - r_j|$ and $R \cdot r_j < 0$.

IV. Selective Neuron Re-Computation (SNRC)

Since the *sigmoid* function always generates a value from 0 to 1 (i.e. (0, 1)), the incorrect value of the neuron in the hidden layer (caused by computational errors) also belongs to (0, 1) (incorrect values exceeding the scope are marked as invalid in the processor); this is the case for Type 1 errors occurring from all three sources (i.e., errors a), errors b) and errors c)). Therefore, a change of the neuron in the hidden layer affected by Type 1 errors is limited in the worst case to 1 (but excluding 1 as value). Define R as the result obtained in step i of the computation for the output neuron (i.e., the value of the exponential term in Eq. (3) with no sign, so as in Eq. (4)).

$$R = \sum_{j=1}^{n} w_j^2 \cdot h_j + b_2 \qquad (4)$$

Then, the deviation from the value in the error-free case for the incorrect R under Type 1 errors must be limited in magnitude to the largest absolute weight of neurons in the hidden layer $|w_{max}^2|$; hence, the magnitude of the error is in the range $(0, |w_{max}^2|)$.

Based on this observation, a method similar to the RBR scheme of [35] can be applied when computing the output of a neuron to determine if either the classification result is reliable, or some parts must be selectively recomputed. For example, the following constraints can be checked to detect the correctness of the classification result in the presence of Type 1 errors:

1) $|R| > |w_{max}^2|$: since the magnitude change of R due to Type 1 errors is limited to $(0, |w_{max}^2|)$, then the value of R can be changed bidirectionally with a range from 0 to w_{max}^2. Therefore, when the condition $|R| > |w_{max}^2|$ is

met, the value of R always keeps the same sign in the presence of Type 1 errors; as illustrated in Figure 4 a), the classification result is reliable.

2) $|R| = |w_{max}^2|$: in this case, the worst-case Type 1 errors make the value of R to approach 0 while maintaining the same sign; this is illustrated in Figure 4 b). Therefore, the classification result is still correct.
3) $|R| < |w_{max}^2|$: in this case, the worst-case Type 1 errors can modify R with a value larger than $|R|$, changing the sign of R; therefore, the classification can be affected by errors. However, further checking can be performed to identify the correctness of the classification results. Similar to the RBR scheme of [35], define r as the sum term of R, i.e., $r = \sum_{j=1}^{n} w_j^2 \cdot h_j$ and r_j as the j^{th} term of the sum $r_j = w_j^2 \cdot h_j$. For the j^{th} neuron in the hidden layer, if R and r_j have the same sign (i.e., $R \cdot r_j \geq 0$) and $|R| \geq |r_j|$, then Type 1 errors that change the value of h_j in the worst case, decrease R with a magnitude of $(0, r_j)$ and thus they cannot change the sign of R (as illustrated in Figure 4 c)). If R and r_j have different signs (i.e., $R \cdot r_j < 0$) and $|R| \geq |w_j^2 - r_j|$, Type 1 errors still have no impact on the sign of R, because its value can only be changed with a magnitude of $(0, w_j^2 - r_j)$ in the worst case; this is illustrated in Figure 4 d). Therefore, when R and r_j meet any of these two constraints, the classification results can be identified as reliable under Type 1 errors.

Algorithm 1 Proposed SNRC scheme

1: Compute Eqs. (1)-(3) to obtain the output *y* and then the classification result *C*;
2: Keep *h*, *R* and *r* = *w*h*;
3: Find the maximum weight w_{max}^2 for neurons in the hidden layer;
4: **if** $|R| \geq |w_{max}^2|$
5: Do re-computation for the output neuron to obtain *y'* and then *C'*;
6: **if** *C* = *C'*
7: Output C;
8: **else**
9: Do re-computation for the output neuron again to obtain *y"* and *C"*;
10: Do majority voting among *C, C'* and *C"*;
11: Output the correct classification result;
12: **end**
13: **else**
14: **for** *j* = 1 to *n* (# of neurons in the hidden layer)
15: **if** $|R| \geq |w_j^2|$
16: Output h_j;
17: **elseif** $(R \cdot r_j > 0$ and $|R| \geq |r_j|)$ or $(R \cdot r_j < 0$ and $|R| \geq |w_j^2 - r_j|)$
18: Output h_j;
19: **else**
20: Do re-computation for j^{th} neuron in the hidden layer to obtain h_j';
21: **if** $h_j = h_j'$;
22: Output h_j;
23: **else**
24: Do re-computation for j^{th} neuron in the hidden layer again to obtain h_j'';
25: Do majority voting among h_j, h_j', h_j'';
26: Output the correct h_j;
27: **end**
28: **end**
29: **end**
30: Compute Eq. (3) twice by using the correct *h*;
31: Compare and vote if needed as per steps 6-11 to output the correct classification result;
32: **End**

As introduced previously, Type 2 errors occur during the mapping from the neurons in the hidden layer to the neuron in the output layer; they originate from the three cases of Figure 3 (but for the output layer neuron), so changing the value of the output. These Type 2 errors can be protected by utilizing a traditional TR scheme; by implementing Eq. (3) for a second (third) time, Type 2 errors can be detected (corrected) based on comparison (majority voting). Since the computation for the neurons in the hidden layer always accounts for the bulk of the implementation of an MLP, then the execution resources incurred by TR are reduced (only used for the output layer neuron).

Since there is no need to detect and correct errors when the classification result is identified as reliable, the so-called Selective Neuron Re-Computation (SNRC) approach is proposed by using Algorithm 1. By determining the cases in which Type 1 errors may change the classification result, the SNRC scheme only introduces re-computation for selective neurons in the hidden layer to detect and correct the errors and ensure that the inputs to the last neuron in the output layer will produce the correct result. Then, the computation for the output layer neuron is repeated to handle Type 2 errors, outputting the same class predicted by the ANN in absence of errors. This significantly reduces the computational overhead compared with traditional protection solutions such as TR when applied to all neurons. The advantage of the proposed SNRC approach in terms of computational resources is evaluated in the next section.

V. Evaluation

In this section, different metrics for an ANN are evaluated by utilizing several widely used datasets from a public repository (details are presented in Table 1) [36]. The datasets have been selected to cover a wide range of applications with different numbers of elements, features, and classification accuracy, so that the results are representative for most scenarios and the proposed scheme can be widely used. Initially, the classification accuracy by using an ANN is assessed; it is also compared with other ML classifiers, in particular, *k*NNs and SVM. Then, the benefits of the proposed SNRC scheme when used to protect an ANN against computational errors are assessed. Simulations are performed by using Matlab, and results are summarized in the following subsections.

A. Classification Performance

*k*NNs is one of the simplest ML classifiers; it is also called lazy learning, because no model needs to be trained. When performing a classification, the distance from the element being classified to every element in the dataset is computed to select

TABLE 1
Datasets Evaluated for Classification

Dataset	Application	# Elements	# Features
Pima Indians diabetes	Medicine	768	8
EEG eye state	Medicine	14980	14
Sonar [37]	Physics	208	60
Ionosphere [38]	Physics	351	34
Electrical grid stability simulated data [39]	Electricity	10000	13
Banknote authentication	Business	1372	4
Qualitative bankruptcy [40]	Business	250	6
Phishing websites [41]	Computer	2456	30
Climate model simulation crashes [42]	Physics	540	17
Cervical cancer (risk factors) [43]	Medicine	858	36
Statlog (German credit data)	Business	1000	20

the neighbors. Once the set of *k*NNs is determined, a majority voting is performed among the classes of the neighbors to predict the class for the new element. Instead, the classification procedure when using an SVM is more complex; a decision hyperplane between the training data with different classes is established first and several support vectors are selected to pursue a maximum margin in the hyperplane. For the datasets that are not linearly separable, kernel functions (e.g, the most widely used radial basis function (RBF)) are utilized to map the original feature space to a higher dimensional one. Therefore, an SVM tends to achieve in most cases a higher classification accuracy than a *k*NNs, because the margin of the hyperplane between different classes can tolerate some noisy elements. When using an ANN, the features of the element are provided as inputs to the network and then activated by using the activation function in the next layer; the neurons in the hidden layer can be considered as providing a refinement of the original features, thus the feature space is also increased, leading to a better classification accuracy.

To compare different classifiers, for each dataset, 70% of the elements are selected randomly to generate the classifier (i.e., the training) and the remaining 30% are used for testing the performance of the classifier (i.e., the testing). Before training the model, the commonly used 10-fold cross-validation

TABLE 2
Performance of ANN, SVM, and *k*NNs for Different Datasets

Dataset	ANN				SVM						*k*NNs			
	n	**CA**	**AA of TP&TN**	**F1-score**	***C***	***γ***	**# SVs**	**CA**	**AA of TP&TN**	**F1-score**	***k***	**CA**	**AA of TP&TN**	**F1-score**
Pima Indians diabetes	17	79.57%	74.18%	77.97%	$2^{0.1}$	$2^{-6.2}$	322	78.70%	72.91%	77.14%	19	76.52%	70.35%	75.23%
EEG eye state	37	73.68%	73.10%	74.61%	$2^{12.0}$	$2^{-1.0}$	3253	63.95%	65.14%	68.99%	3	69.45%	69.20%	69.98%
Sonar	126	93.55%	93.31%	93.55%	$2^{2.2}$	$2^{-5.1}$	125	91.94%	91.38%	92.06%	3	85.48%	84.90%	86.15%
Ionosphere	146	95.28%	91.07%	91.80%	$2^{4.5}$	$2^{-4.0}$	108	93.40%	95.51%	95.30%	11	87.74%	82.14%	84.85%
Electrical grid stability simulated data	30	99.77%	99.72%	99.72%	$2^{10.0}$	2^{-10}	241	99.47%	99.28%	99.28%	19	94.20%	92.62%	93.04%
Banknote authenti-cation	11	100.00%	100.00%	100.00%	$2^{0.1}$	$2^{-3.0}$	103	100.00%	100.00%	100.00%	7	100.00%	100.00%	100.00%
Qualitative bankruptcy	29	100.00%	100.00%	100.00%	$2^{0.1}$	$2^{-3.4}$	66	100.00%	100.00%	100.00%	3	98.67%	98.65%	98.67%
Phishing websites	91	95.52%	95.69%	95.64%	$2^{5.0}$	$2^{-5.0}$	338	94.84%	94.97%	94.93%	5	92.81%	92.91%	92.86%
Climate model simulation crashes	84	98.15%	92.21%	92.69%	$2^{5.2}$	$2^{-8.6}$	72	98.77%	99.35%	99.35%	5	96.30%	77.60%	81.41%
Cervical cancer (risk factors)	139	92.64%	59.37%	70.85%	$2^{4.0}$	$2^{-9.0}$	120	92.64%	52.50%	67.80%	5	92.25%	50.00%	66.67%
Statlog (German credit data)	61	75.67%	63.11%	71.02%	$2^{3.9}$	$2^{-9.0}$	382	73.00%	64.39%	70.00%	9	72.00%	59.04%	68.04%
Average	-	91.10%	85.61%	87.99%	-	-	-	89.70%	85.04%	87.71%	-	87.77%	79.76%	83.35%

TABLE 3
Percentage of Errors that Change the Classification Results

Dataset	Unprotected ANN	Unprotected SVM	Unprotected *k*NNs
Pima Indians diabetes	10.66%	10.12%	1.81%
EEG eye state	35.02%	49.70%	6.79%
Sonar	3.19%	17.33%	8.21%
Ionosphere	1.28%	11.35%	1.90%
Electrical grid stability simulated data	3.18%	48.44%	0.97%
Banknote authentication	8.66%	0.82%	0
Qualitative bankruptcy	2.11%	1.67%	0.77%
Phishing websites	1.68%	18.59%	1.49%
Climate model simulation crashes	0.93%	40.96%	2.20%
Cervical cancer (risk factors)	0.80%	36.54%	0.01%
Statlog (German credit data)	4.96%	39.95%	2.17%
Average	6.59%	25.04%	2.39%

methodology is used to select the parameters of the classifiers to achieve a good performance, i.e., the number of nearest neighbors k in a kNNs, the penalty parameter C and the kernel parameter γ in an SVM, and the number of neurons n in the hidden layer in the MLP considered in this paper. The training dataset is split into 10 subsets of equal size first and then, different values of the parameters are checked to find the best cross-validation accuracy. When choosing the parameters, the candidate values for k in a kNNs are selected as odd numbers from 3 to 19, for C and γ in an SVM the various pairs of (C, γ) values from $C = 2^{-5}, 2^{-4}, \ldots, 2^{15}$ and $\gamma = 2^{-15}, 2^{-14}, \ldots, 2^{-3}$ or a close range as suggested in [44], and for n in the MLP, 2-6 times the number of features (i.e. the number of neurons in the input layer) as proposed in [45].

For each dataset given in Table 1, three widely used classification related metrics, including the classification accuracy (CA) found by employing the trained parameters (i.e., n, C, γ, and k), the average accuracy of true-positive and true-negative (denoted as AA of TP&TN) and the F1-score, are presented in Table 2; the results confirm that on average an ANN provides a better classification performance than SVM and kNNs as discussed previously.

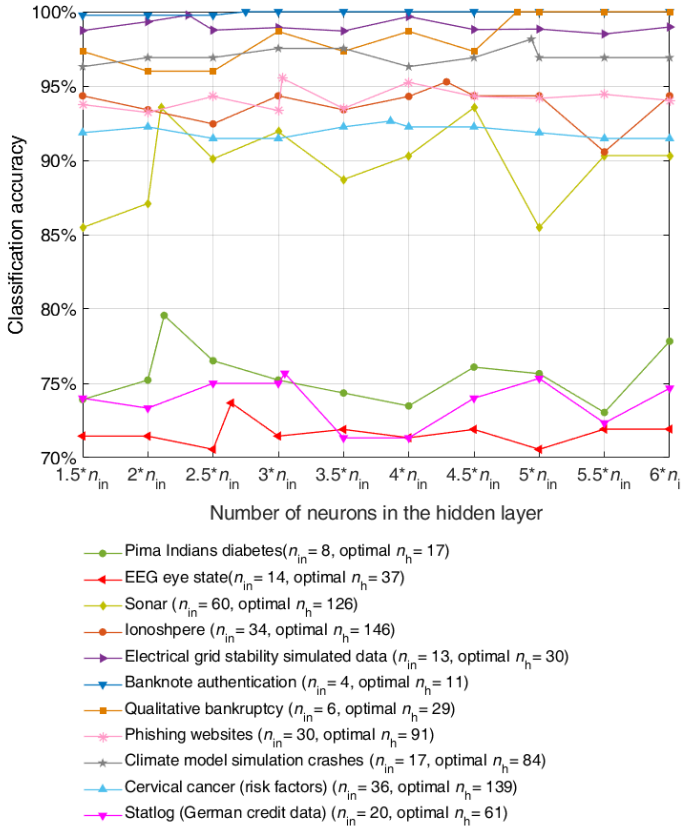

Figure 5 Classification accuracy for different number of neurons in the hidden layer (results for the optimal number of neurons are also included).

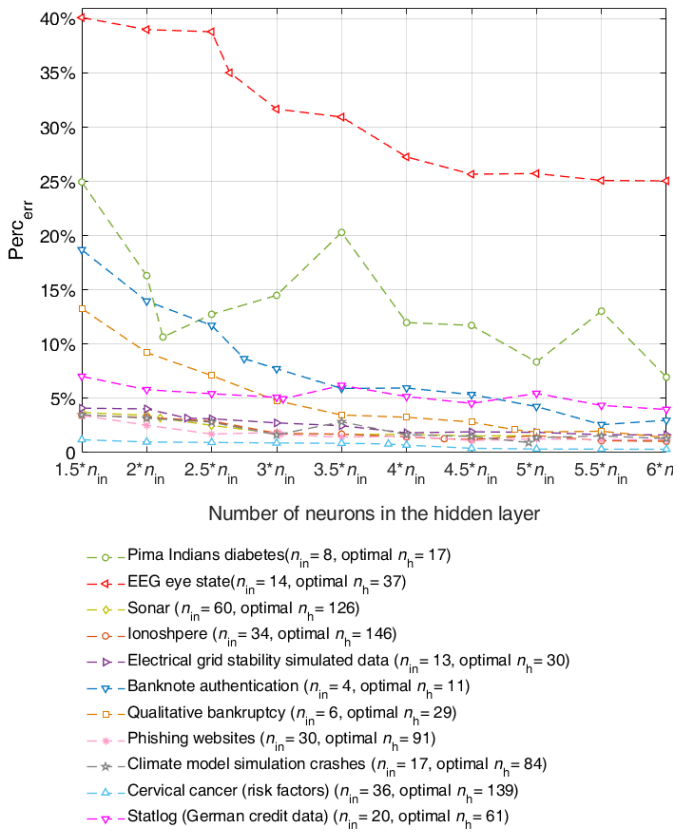

Figure 6 Percentage of errors that modify the classification results ($Perc_{err}$) for different number of neurons in the hidden layer (results for the optimal number of neurons are also included).

B. Robustness Under Computational Errors

Prior to analyzing the effectiveness of the proposed scheme, the robustness of the unprotected classifiers in the presence of errors is assessed first (where a classifier is said to be robust if errors have no impact on the classification result). Since the distance computation (kernel computation) accounts for the bulk of the *k*NNs (SVM) implementation, computational errors modifying a distance (the kernel result for a support vector) are considered; for an ANN, the impact of Type 1 and Type 2 errors (affecting the value of one neuron in the hidden layer or the output layer as discussed in Section III) are studied. Such errors are injected in the classifiers as detailed next; the process of each error injection is repeated 100,000 times to calculate the average value. Furthermore, the following features are considered for the errors:

Error position: errors are assumed to be uniformly distributed, i.e., each distance computed in a *k*NNs, each kernel result computed in an SVM, and each neuron value computed in an ANN have the same probability of error occurrence.

Error magnitude: in a *k*NNs, the value of the incorrect distance is set to a random value within the scope range of (0, $d_{\max}$] where $d_{\max}$ denotes the largest value of all computed distances; this permits to deal with all possible cases, because an element with an incorrect distance larger than $d_{\max}$ has the same impact as an element with a distance of $d_{\max}$. Similarly, the value of the incorrect kernel result in an SVM (with the RBF kernel considered) is set to a random value within (0, 1], which is the valid range of the RBF kernel result; the value of the incorrect neuron in an ANN considered in this paper is also set to a random value within (0, 1), because it is within the possible range of the *sigmoid* function.

Table 3 presents the percentage of errors that change the classification results in different classifiers; these results show that the unprotected *k*NNs is more robust than the SVM in all cases. This occurs, because in an SVM, the classification result is determined by the classes of the selected support vectors; if the kernel result for one support vector is affected by errors, then the final classification result has a specific probability to be changed. Instead, the classification result obtained with a *k*NNs depends on the classes of the neighbors. Since the number of neighbors k is always significant smaller than the number of elements in the datasets, distance modifying errors have a low probability of affecting a neighbor. In an ANN, as discussed previously, the features of the element are provided as inputs to the network and then activated by using the activation function in the next layer, thus the neurons in the hidden layer can be considered as a refinement of the original features. In this case, errors that change the value of a neuron, can be considered as a change of a "feature", so moving the element in a dimension of the feature space. Therefore, the impact of errors on the classification result is related to the number of neurons in the hidden layer, as well as performance in terms of accuracy. This has been verified and shown in Figures 5 and 6, that plot the classification and the percentage of errors that modify the classification results (i.e., Perc_{err} for different number of neurons in the hidden layer. As per Figures 5 and 6, the impact of errors tends to be smaller with a larger number of neurons for all cases (i.e, Perc_{err} decreases); moreover, for the datasets with a lower classification accuracy

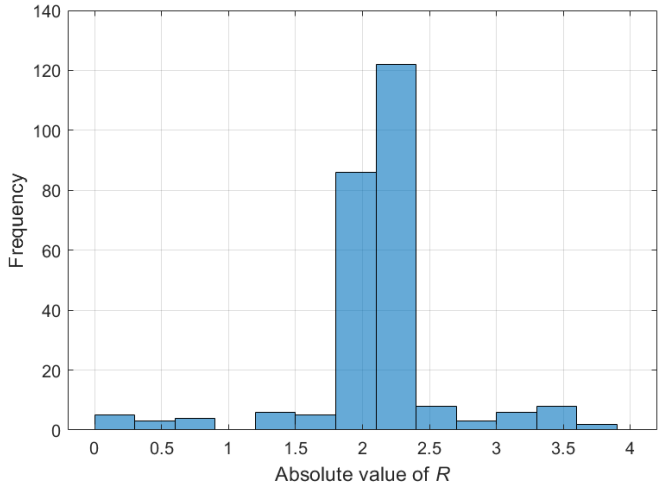

Figure 7 Distribution of $|R|$ for the Cervical cancer (risk factors) dataset with $|w^2_{max}|$ = 1.08 and $|w^2_{min}|$ = 4E-4.

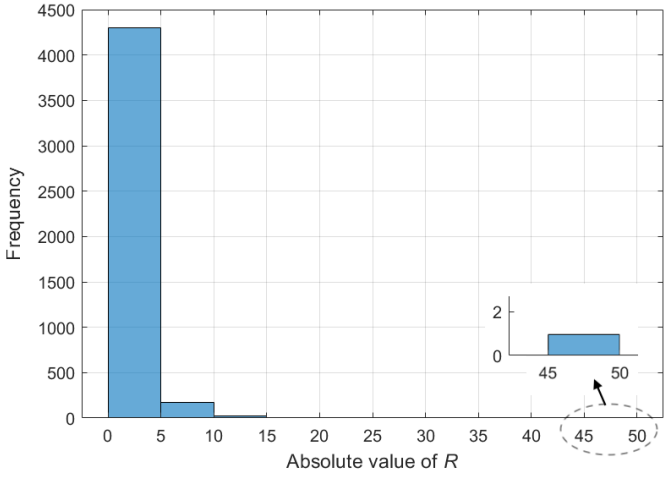

Figure 8 Distribution of $|R|$ for the EEG eye state dataset with $|w_{max}|$ = 129.08 and $|w^2_{min}|$ = 0.05.

(e.g, <80%), such as the Pima Indian diabetes, the impact of errors shows an additional trend, namely Perc_{err} decreases when the accuracy is higher.

C. Re-computation Overhead

When employing the proposed SNRC scheme to protect an ANN, errors are analyzed to ensure that the predicted class is reliably correct. As discussed before, SNRC can detect the cases in which the classification result is not changed by errors. This makes SNRC more effective compared to a traditional TR-based technique, which always repeats the entire computation process, so introducing a large overhead such as 100% re-computation overhead for detection.

SNRC is assessed for the datasets of Table 1; the percentage of elements for which re-computation for some neurons is required, is presented in Table 4. As per Table 4, the SNRC scheme saves more than 60% re-computation overhead for all considered datasets (in particular, more than 90% for most datasets); this occurs, because these datasets either have for most elements an absolute R (as per Eq. (4)) larger than the

TABLE 4
Percentage of Elements for which a Re-computation Operation in ANN, SVM and *k*NNs can be Saved

Dataset	ANN with SNRC							SVM with RBR	*k*NNs with VM
	$\|R\| \in [\|w_{max}\|, +\infty)$		$\|R\| \in [\|w_{min}\|, \|w_{max}\|)$		$\|R\| \in (-\infty, \|w_{min}\|)$		Average of neurons		
	% elements	% Neurons	% elements	% neurons	% elements	% neurons			
Pima Indians diabetes	13.91%	100.00%	76.09%	76.40%	10.00%	15.69 %	**73.61%**	**70.82%**	**90.75%**
EEG eye state	0	100.00%	96.97%	65.93%	3.03%	0	**63.93%**	**7.29%**	**64.89%**
Sonar	53.23%	100.00%	46.77%	82.81%	0	0	**91.96%**	**64.14%**	**80.10%**
Ionosphere	78.30%	100.00%	21.70%	84.51%	0	0	**96.64%**	**73.80%**	**91.69%**
Electrical grid stability simulated data	0	100.00%	100.00%	97.79%	0	0	**97.79%**	**12.44%**	**96.50%**
Banknote authentication	4.85%	100.00%	95.15%	92.74%	0	0	**93.09%**	**95.58%**	**100.00%**
Qualitative bankruptcy	92.00%	100.00%	8.00%	93.24%	0	0	**99.46%**	**92.16%**	**97.31%**
Phishing websites	86.02%	100.00%	13.98%	83.43%	0	0	**97.68%**	**55.04%**	**90.32%**
Climate model simulation crashes	72.84%	100.00%	27.16%	96.21%	0	0	**98.97%**	**21.20%**	**91.63%**
Cervical 9cancer (risk factors)	95.35%	100.00%	4.65%	87.17%	0	0	**99.40%**	**29.85%**	**97.66%**
Statlog (German credit data)	44.00%	100.00%	55.67%	84.04%	0.33%	1.64%	**90.78%**	**23.73%**	**83.20%**
Average	-	-	-	-	-	-	**91.21%**	**49.64%**	**89.46%**

absolute maximum weight in the hidden layer (corresponding to steps 4-7 in Algorithm 1 in Section III), or meet the constraints for *r* to determine a reliable neuron (corresponding to steps 15-18 in Algorithm 1). For example, for the Cervical cancer (risk factors) dataset with $|w_{max}^2|$ = 1.08 and $|w_{min}^2|$= 4E-4, the distribution for $|R|$ is plotted in Figure 7; it has an absolute value larger than $|w_{max}^2|$ in most cases. Instead, for the datasets with a low reduction of re-computation, such as the EEG eye state dataset with $|w_{max}^2|$ = 129.08 and $|w_{min}^2|$= 0.05, the distribution for $|R|$ is given in Figure 8; in this case, the savings come only from the smaller weights, because the largest weight is always greater than $|R|$.

A further observation is that the effectiveness of SNRC is related to the robustness of the unprotected ANN; for datasets

TABLE 5
Comparison in Number of Re-computation Operations and Power Dissipation for Different Schemes

Dataset	Traditional TR					Proposed SNRC							
	Add	Mul	Div	Exp	Power (mW)	Add	Sub	Abs	Mul	Div	Exp	Comp	Power (mW)
Pima Indians diabetes	131328	131328	13824	13824	41.10 (***100%***)	44831	1343	12814	46175	4213	4213	24334	14.71 (***35.79%***)
EEG eye state	8883140	8883140	569240	569240	2263.06 (***100%***)	3567871	0	567426	3567871	214889	214889	1091726	941.91 (***41.62%***)
Sonar	1625104	1625104	26416	26416	304.17 (***100%***)	154947	322	12996	155269	2315	2315	38788	30.33 (***9.97%***)
Ionosphere	1845207	1845207	51597	51597	375.90 (***100%***)	111886	2946	14768	114832	2074	2074	65312	24.11 (***6.41%***)
Electrical grid stability simulated data	4510000	4510000	310000	310000	1178.63 (***100%***)	402820	0	320000	402820	16630	16630	600000	118.78 (***10.08%***)
Banknote authenti-cation	91924	91924	16464	16464	38.36 (***100%***)	21677	0	17104	21677	2415	2415	29452	8.46 (***22.05%***)
Qualitative bankruptcy	58250	58250	7500	7500	20.16 (***100%***)	7774	93	1173	7868	289	289	7923	1.96 (***9.70%***)
Phishing websites	7154328	7154328	225952	225952	1494.06 (***100%***)	386447	5443	41599	391890	7633	7633	260183	83.68 (***5.60%***)
Climate model simulation crashes	862380	862380	45900	45900	206.47 (***100%***)	54305	293	13692	54597	1007	1007	57972	12.48 (***6.04%***)
Cervical cancer (risk factors)	4532814	4532814	120120	120120	914.04 (***100%***)	146446	0	7262	146446	1570	1570	124808	29.83 (***3.26%***)
Statlog (German credit data)	1343000	1343000	62000	62000	308.14 (***100%***)	179974	4682	40641	184656	6618	6618	99641	43.98 (***14.27%***)
Average	2821589	2821589	131728	131728	649.46 (***100%***)	461725	1375	95407	463100	23605	23605	218194	119.11 (***18.34%***)

with lower percentage of errors that modify the classification result (as per Table 3), the SNRC technique achieves larger re-computation savings. This is also the case for different classifiers; ANN and *k*NNs are more robust than SVM, and the average re-computation savings in an ANN protected by the SNRC technique and in a *k*NNs protected by the VM technique [33] are larger than for a SVM protected by the RBR technique [35] (as shown in Table 4).

D. Power Dissipation

Since computational operations are related to the performance of the processor, the power dissipations for re-computation required by the traditional TR-based protection and the proposed SNRC technique are evaluated and compared next.

Initially, Table 5 shows the number of operations required by the different schemes, including addition, subtraction, absolute value, multiplication, division, exponentiation and comparison. Then, the power dissipated for performing these operations in an open source processor [46] are evaluated by using Synopsys PrimeTime and mapping the hardware for these operations to a 7 nm technology library; the results are given in Table 6.

TABLE 6
Power dissipation required for Different Operations

Operation	Power (mW)
Addition	2.26E-5
Subtraction	2.25E-5
Absolute value	4.21E-5
Multiplication	14.16E-5
Division	22.65E-5
Exponentiation	118.67E-5
Comparison	2.61E-5

Next, the power dissipation for the different schemes is found by combining the number of operations (as per Table 5) with their power dissipation (as per Table 6). Results for each dataset considered are also shown in Table 5; even though the proposed SNRC technique introduces some operations to perform the comparison (as discussed previously), the power saving in terms of re-computation is significantly large. This is because, the saved operations (for example exponentiation) are complex while the introduced operations (mostly comparison or subtraction) are simple and thus the savings are significantly larger that the incurred overhead. As per Table 5, the SNRC can reduce 81.66% power dissipation on average for the datasets considered.

VI. Extending SNRC to ANNs with Other Functions

As discussed before, another widely used activation function for the hidden layer in an ANN is the *tanh* function; it is an extended version of the *sigmoid* and is given by Eq. (5).

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = 2 \cdot sigmoid(2x) - 1 \qquad (5)$$

Since the *tanh* function generates a similar shape as the *sigmoid* function, then these functions achieve a similar classification accuracy; this has been verified in Table 7 by utilizing the same datasets, where *n* is the number of neurons in the hidden layer (note that as the activation function has been changed, then the number of neurons in the hidden layer determined during the training process may be different). However, as per Eq. (5), the *tanh* function generates a larger value range of (-1, +1), this makes the network to have a slower convergence saturation speed and thus, a higher training efficiency when using the BP algorithm.

The proposed SNRC technique is also applicable for an ANN using *tanh* as activation function. As per similar reasoning to one for the *sigmoid* function, computational errors that modify the value of a neuron in the hidden layer, can only change the value of R (as Eq. (4)) with a limited magnitude of $(-|w_{max}^2|,+|w_{max}^2|)$. Therefore, the classification result can be identified as reliable when any of the following constraints is met.

1) $|R| > |w_{max}^2|$: the reason is the same as for the case in which the *sigmoid* function is used; the sign of R does not change by Type 1 errors, so the classification result is reliable.
2) R and $r_j = w_j^2 \cdot h_j$ have the same sign and $|R| \geq |w_j^2| + |r_j|$: in this scenario, the sign of R also remains the same as the error-free case, because Type 1 errors change the value of a neuron with a worst-case magnitude of $|w_j| + |r_j|$.

TABLE 7
Results for ANN with Different Activation Functions

Dataset	*n*		Highest accuracy		$Perc_{err}$		Re-computation saving	
	sigmoid	*tanh*	*sigmoid*	*tanh*	*sigmoid*	*tanh*	*sigmoid*	*tanh*
Pima Indians diabetes	17	17	79.57%	79.57%	10.66%	14.31%	73.61%	77.34%
EEG eye state	37	37	73.68%	74.25%	35.02%	27.63%	63.93%	38.02%
Sonar	126	126	93.55%	91.94%	3.19%	3.69%	91.96%	93.00%
Ionosphere	146	146	95.28%	96.23%	1.28%	1.10%	96.64%	98.64%
Electrical grid stability simulated data	30	30	99.77%	99.80%	3.18%	3.62%	97.79%	98.84%
Banknote authentication	11	11	100.00%	100.00%	8.66%	5.27%	93.09%	95.50%
Qualitative bankruptcy	29	29	100.00%	100.00%	2.11%	2.77%	99.46%	98.53%
Phishing websites	91	91	95.52%	95.66%	1.68%	1.94%	97.68%	97.03%
Climate model simulation crashes	84	84	98.15%	98.15%	0.93%	1.02%	98.97%	98.93%
Cervical cancer (risk factors)	139	139	92.64%	92.64%	0.80%	0.50%	99.40%	99.85%
Statlog (German credit data)	61	60	75.67%	75.67%	4.96%	3.65%	90.78%	94.45%
Average	-	-	91.26%	91.26%	6.59%	5.95%	91.21%	90.01%

3) R and r_j have different signs and $|R| \geq |w_j - r_j|$: same as the case in which the *sigmoid* function is used, errors in this scenario have no impact on the classification result.

Therefore, to implement the SNRC scheme for an ANN with the *tanh* function, Algorithm 1 can be employed by simply modifying step 17 to "**elseif** ($R \cdot r_j > 0$ and $|R| \geq |w_j| + |r_j|$) or ($R \cdot r_j < 0$ and $|R| \geq |w_j - r_j|$)".

The robustness of the unprotected ANN with the *tanh* function and the efficiency by employing the SNRC scheme to protect the network, are evaluated by using the same error injection method as in the previous section, but with an error magnitude of (-1, +1). As per the results presented in Table 7, the percentage of errors that modify the classification results (i.e., $Perc_{err}$) of the unprotected ANN with the *tanh* function, is on average slightly lower than for the *sigmoid* function; the SNRC scheme also works efficiently by achieving an average reduction of re-computation with more than 90%, even if it is slight lower than using the *sigmoid* function.

It is important to note that the proposed SNRC technique is applicable to all activation functions that have bounded values (e.g., *sigmoid* and *tanh*). However, it cannot be used for activation functions that are not bounded, because in such case, the classification result cannot be determined to be reliable; this occurs, because an error modifying a single neuron computation can generate a large deviation in the final result. Therefore, SNRC is not applicable for example, to *ReLU* as an activation function that is usually employed for deeper ANNs (so having more than one hidden layer) due to its training efficiency, because it can take values from zero to positive infinity. However, ANNs with a more complex structure tend to be more error resilient than an ANN with only one hidden layer, because there are more neurons and errors changing a neuron tend to have a smaller impact on the classification result.

A further observation is that complex ANNs can be efficiently protected with Symptom-based Error Detection (SED) [9]; SED keeps track of the maximum observed value at the output of a neuron during training and uses it with some margin to trigger error detection when using the ANN. Therefore, if the maximum value is exceeded during the inference process, a potential error is detected. Interestingly, SED is not effective when *sigmoid* or *tanh* functions are used, because the output values under normal operation tend to be close to the maximum due to the saturation of these functions for large input values. This has been verified by simulation; the results are shown in Table 8. Table 8 gives the percentage of detectable errors modifying the classification results when using SED to protect an MLP with the *sigmoid*/*tanh* function. As per Table 8, only less than 20% errors on average can be detected; therefore, SED is best suitable for deeper ANNs, but not for MLP and the datasets considered in this paper.

TABLE 8
Percentage of Errors Modifying the Classification Results that can be Detected by Using Symptom-based Error Detection (SED)

Dataset	*sigmoid*	*Tanh*
Pima Indians diabetes	16.10%	36.04%
EEG eye state	15.20%	25.24%
Sonar	20.85%	34.62%
Ionosphere	10.32%	10.33%
Electrical grid stability simulated data	0	0.04%
Banknote authentication	0.06%	1.65%
Qualitative bankruptcy	0.07%	5.94%
Phishing websites	0.99%	0.75%
Climate model simulation crashes	1.42%	4.97%
Cervical cancer (risk factors)	40.00%	79.78%
Statlog (German credit data)	8.21%	10.04%
Average	10.29%	19.04%

VII. Conclusion

This paper has presented an efficient error-tolerant technique for protecting Artificial Neural networks (ANNs) against computational errors in the neurons. Errors that modify the value of the neurons can change the predicted class when the ANN is used to perform a classification task. Therefore, when ANNs are used in many safety-critical applications and implemented in resource-constrained platforms, efficient protection schemes are needed. This has motivated the design of the proposed Selective Neuron Re-Computation (SNRC) technique by exploiting the inherent redundancy of an ANN.

As per the magnitude of the errors in an ANN with the *sigmoid* or *tanh* as activation function, SNRC performs several checks on the internal computation results and the weights in the ANN implementation to identify the correctness of the classification result. In particular, the neurons for which an error can affect the final classification outcome, are readily identified; then, the re-computation operations are only executed for these neurons, so significantly reducing the re-computation overhead.

The impact of computational errors in an ANN with a single hidden layer and using *sigmoid* as activation function has been evaluated using a wide range of datasets; results have shown that the percentage of errors that modify the classification results, is up to 35.02%. The effectiveness of the SNRC has also been assessed for the considered datasets; results have shown that compared with the current temporal redundancy-based protection, SNRC significantly reduces the protection overhead (in particular, saving more than 60% re-computation for the neurons in all cases and 90% for most of the datasets, which translates to 81.66% saving in power dissipation on average). The case of employing SNRC to an ANN with the *tanh* function has also been analyzed and evaluated; it can also achieve a significant reduction for re-computation. Overall, the proposed SNRC technique efficiently protects ANNs with *sigmoid/tanh* as activation functions (so usually shallow ANNs).

References

[1] S. Das, A. Dey, A. Pal, *et al.*, "Applications of Artificial Intelligence in Machine Learning: Review and Prospect", International Journal of Computer Applications, vol.115, no.9, pp.31-41, 2015.

[2] C. Szegedy, W. Liu and Y. Jia, "Going Deeper with Convolutions, Computing Research Repository," in Proc of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778. 2016.

[4] E. Khadangi, A. Bagheri, “Comparing MLP, SVM and KNN for Predicting Trust Between Users in Facebook”, in ICCKE, pp. 466-470, 2013.

[5] C. Deng, S. Liao, B. Yuan, “PERMCNN: Energy-Efficient Convolutional Neural Network Hardware Architecture with Permuted Diagonal Structure”, IEEE Transactions on Computers, 2020 (Early Access).

[6] E. Nurvitadhi, D. Sheffield, J. Sim, et al, “Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC”, in IEEE International Conference on Field-Programmable Technology (FPT), pp. 77-84, 2016.

[7] C. Wang, L. Gong. X. Ma, et al, “WooKong: A Ubiquitous Accelerator for Recommendation Algorithms with Custom Instruction Sets on FPGA”, IEEE Transactions on Computers, vol. 69, no. 7, pp. 1071-1082, 2020.

[8] X. Wang, M. Magno, L. Cavigelli, et al, "FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things," in IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4403-4417, 2020.

[9] G. Li, H. Siva and S. Michael, “Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications,” in Proc of the International Conference for High-Performance Computing, Networking, Storage and Analysis (SC), 2017.

[10] P. Koopman, M. Wagner, “Autonomous Vehicle Safety: An Interdisciplinary Challenge”, IEEE Intelligent Transportation Systems Magazine, vol. 9, no. 1, pp. 90-96, 2017.

[11] N. Kanekawa, E.H. Ibe, T. Suge, et al., “Dependability in Electronic Systems”, Springer Science & Business Media, New York, 2011.

[12] J. Marques, J. Andrade and G. Falcao, “Unreliable Memory Operation on a Convolutional Neural Network Processor,” in Proc of the IEEE International Workshop on Signal Processing Systems. Lorient. France, 2017.

[13] D. S. Phatak and I. Koren, “Complete and Partial Fault Tolerance of Feedforward Neural Nets,” IEEE Transactions on Neural Networks, vol. 6, no. 2, pp. 446-456, 1995.

[14] M. D. Emmerson and R. I. Damper, “Determining and Improving the Fault Tolerance of Multilayer Perceptrons in a Pattern-Recognition Application,” IEEE Transactions on Neural Networks, vol. 4, no. 5, pp. 788-793, 1993.

[15] J. L. Bernier, J. Ortega and I. Rojas, “Improving the Tolerance of Multilayer Perceptrons by Minimizing the Statistical Sensitivity to Weight Deviations," Neurocomputing, vol. 31, pp. 87-103, 2000.

[16] S. Cavalieri and O. Mirabella, “A Novel Learning Algorithm Which Improves the Partial Fault Tolerance of Multilayer Neural Networks,” Neural Networks, vol. 12, no. 1, pp. 91-106, 1999.

[17] B. S. Arad and A. El-Amawy, “On Fault Tolerant Training of Feedforward Neural Networks,” Neural Networks, vol. 10, no. 3, pp. 539-553, 1997.

[18] I. Oz and S. Arslan, "A Survey on Multithreading Alternatives for Soft Error Fault Tolerance", in ACM Computing Surveys, vol. 52, 2019.

[19] M. Biasielli, C. Bolchini, L. Cassano, et al, “A Neural Network Based Fault Management Scheme for Reliable Image Processing”, IEEE Transactions on Computers, vol. 69, no. 5, pp. 764-776, 2020.

[20] F. Rosenblatt, “The Perceptron: A probabilistic Model for Information Storage and Organization in the Brain”, Psychological Review, vol. 65, no. 6, pp. 386-408, 1958.

[21] Y. Liu, S. Liu, Y. Wang, et al, “A Stochastic Computational Multi-Layer Perceptron with Backward Propagation”, IEEE Transactions on Computers, vol. 67, no. 9, pp. 1273-1286, 2018.

[22] E. A. M. Shenouda, “A Quantitative Comparison of Different MLP Activation Functions in Classification”, in Proceedings of the Third international conference on Advances in Neural Networks” pp. 849–857, 2006.

[23] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo and T. Toba, “Impact of Scaling on Neutron-Induced Soft Error in SRAMs from a 250 nm to a 22 nm Design Rule”, IEEE Transactions on Electron Devices, vol. 57, no. 7, pp. 1527-1538, 2010.

[24] I. Chatterjee, B. Narasimham, N. N. Mahatme, et al, “Impact of Technology Scaling on SRAM Soft Error Rates”, IEEE Transactions on Nuclear Science, vol. 61, no. 6, pp. 3512-3518, 2014.

[25] C. L. Chen and M. Y. Hsiao, “Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review,” in IBM Journal of Research and Development, vol. 28, no. 2, pp. 124-134, 1984.

[26] G. Umanesan, E. Fujiwara, “A Class of Random Multiple Bits in a Byte Error Correcting and Single Byte Error Detecting (S/sub t/b/EC-S/sub b/ED) codes, IEEE Transactions on Computers, vol. 52, vol. 7, pp. 835-847, 2003.

[27] H. Farbeh, L. Delshadtehrani, H. Kim, et al, “ECC-United Cache: Maximizing Efficiency of Error Detection/Correction Codes in Associative Cache Memories”, IEEE Transactions on Computers, 2020 (Early Access).

[28] Intel, “Intel Stratix 10 Embedded Memory User Guide”, [Online] available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-memory.pdf, pp. 1-120, 2019.

[29] Texas Instruments, “DDR ECC Reference Design to Improve Memory Reliability in 66AK2Gx-Based Systems”, [Online] available: http://www.ti.com/lit/ug/tidubo4b/tidubo4b.pdf?ts=1588313107130, pp. 1-13, 2018.

[30] M. Nicolaidis, “Design for Soft Error Mitigation”, IEEE Transactions on Device and Materials Reliability, vol. 5, no. 3, pp. 405-418, 2005.

[31] Y. Ma and H. Zhou, “Efficient Transient-Fault Tolerance for Multithreaded Processors Using Dual-Thread Execution”, in Proceedings of the International Conference on Computer Design, pp. 120-126, 2006.

[32] K. Chen, L. Chen, P. Reviriego and F. Lombardi, “Efficient Implementations of Reduced Precision Redundancy (RPR) Multiply and Accumulate (MAC),” in IEEE Transactions on Computers, vol. 68, no. 5, pp. 784-790, 2019.

[33] S. Liu, P. Reviriego, J.A. Hernández, et al., “Voting Margin: A Scheme for Error-Tolerant *k* Nearest Neighbors Classifiers”, IEEE Transactions on Emerging Topics in Computing, 2019 (Early Access).

[34] S. Liu, P. Reviriego, P. Montuschi, et al, "Error-Tolerant Computation for Voting Classifiers with Multiple Classes," IEEE Transactions on Vehicular Technology, vol. 69, no. 11, pp. 13718-13727, 2020.

[35] S. Liu, P. Reviriego, X. Tang, et al, “Result-Based Re-Computation (RBR) for Error-Tolerant Classification by a Support Vector Machine (SVM)”, IEEE Transactions on Artificial Intelligence, vol.1, no.1, pp. 62-73, 2020.

[36] D. Dua and C. Graff “UCI Machine Learning Repository”, Irvine, CA: University of California, School of Information and Computer Science, 2019.

[37] R. P. Gorman, T.J. Sejnowski, “Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets. Neural networks, vol. 1, no. 1, pp.75-89, 1988.

[38] V. G. Sigillito, S. P. Wing, L.V. Hutton, et al., “Classification of Radar Returns from the Ionosphere Using Neural Networks”, Johns Hopkins APL Technical Digest, vol. 10, no. 3, pp.262-266, 1989.

[39] V. Arzamasov, K. Bohm, P. Jochem, “Towards Concise Models of Grid Stability”, in IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (Smart Grid Comm), pp. 1-6, 2018.

[40] M. J. Kim, I. Han, “The Discovery of Experts' Decision Rules from Qualitative Bankruptcy Data Using Genetic Algorithms. Expert Systems with Applications, vol. 25, no. 4, pp. 637-646, 2003.

[41] R. M. Mohammad, F. Thabtah, L. McCluskey, “Intelligent Rule-based Phishing Websites Classification”, IET Information Security, vol. 8, no. 3, pp. 153-160, 2014.

[42] D. D. Lucas, R. Klein, J. Tannahill, et al., “Failure Analysis of Parameter-Induced Simulation Crashes in Climate Models”, in Geosci. Model Dev. Discuss., vol. 6, pp. 585-623, 2013.

[43] K. Fernandes, J.S. Cardoso and J. Fernandes, “Transfer Learning with Partial Observability Applied to Cervical Cancer Screening”, Iberian Conference on Pattern Recognition and Image Analysis, Springer, Cham, 2017.

[44] C. W. Hsu, C. C. Chang and C. J. Lin, “A Practical Guide to Support Vector Classification” pp.1-16, 2016.

[45] Q. Hu, X. Tang, W. Tang, “A Smart Chair Sitting Posture Recognition System Using Flex Sensors and FPGA Implemented Artificial Neural Network”, IEEE Sensors Journal, vol. 20, no. 14, pp. 8007-8016, 2020.

[46] OpenRisc 1200 HP, Hyper Pipelined OR1200 Core, [Online], available: https://opencores.org/projects/or1200_hp, 2018.