# Detecting hard synapses faults in artificial neural networks

Martin Krcma, Zdenek Kotasek, Jakub Lojda
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno, Czech Republic
Email: ikrcma@fit.vutbr.cz, kotasek@fit.vutbr.cz, ilojda@fit.vutbr.cz

***Abstract*—This paper presents the concepts of detecting hard faults in artificial neural network synapses using the modification of the neural network settings. The core of this work is based on weights values modification and inserting the chosen testing data when comparing the neural network output to the known valid results. The paper also discusses the problem of neural networks output saturation and provides experiments regarding an influence of the neural network settings to the problem.**

## I. Introduction

The artificial neural networks [7] are one of the important models of soft-computing and artificial intelligence. Their structure is inspired by the structure of the human brain and they dispose of a high capability of learning and memorizing to solve various types of tasks. Basically, the goal of the artificial neural network is to learn the relation between two sets of data vectors, to generalize the relation, to determine its features and to use it for the determining the relation of the unknown vectors belonging to the same problem. This capability can be used for classification tasks, for time-series and functional prediction, for control tasks, image recognition, clustering and other tasks.

The networks have been implemented in various kinds of devices starting from analog computers to the most modern processors, VLSI units, graphical processing units and FPGAs. In the hardware implementation there is a chance that a fault occurs in the device influencing its computation. The fault can be transient, temporary which can be solved by numerous ways. If the fault is hard and permanent however, it may not be possible to fix it. In this case, the detecting of the fault is even more important than in the case of temporary faults because the computation of the device and the data it produces are permanently affected by the fault. This paper deals with one of the possible ways how to detect hard faults in neural network synapses.

## II. Artificial Neural Networks

Neural networks are composed of a set of *neurons*. A neuron is a simple unit which computes an *activation function* (2) over a result of a *basis function* which is often a *weighted sum* (1) of the neuron inputs. A neuron is illustrated in Fig. 1. The neurons are interconnected with the weighted connections called *synapses*. The learning of the neural network is basically a process of setting the weights.

978-1-7281-1756-0/19/$31.00 

The value $\theta$ in equation (2) represents the neuron *threshold*. The threshold allows us to affect the shape of the neuron activation function (its position on the $x$ axis) which increases the power of the network and the efficiency of its learning.

The neurons are often organized into the layered structure composed of an input layer, output layer and a number of hidden layers. This type of structure is illustrated in Fig. 2. Sometimes, the neural network is composed of only one or two layers or of layers with different neuron types (i.e. neurons with different basis and activation functions) or interconnection structures.

$$net = \sum_{i=1}^{n} x_i w_i \quad (1)$$

$$y = f(net + \theta) \quad (2)$$

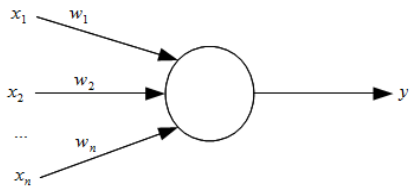


Fig. 1. The neuron.

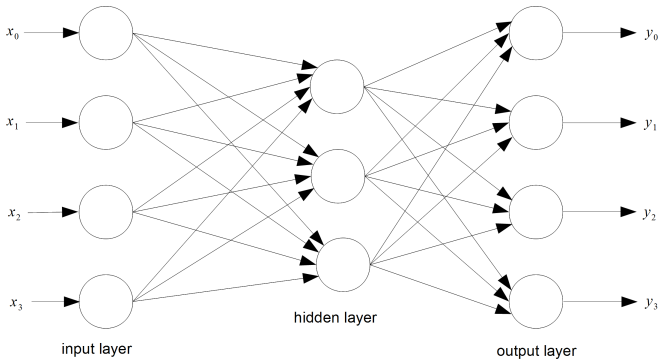


Fig. 2. The neural network layered structure.

### A. Activation functions approximations

A number of different activation functions which are used in neural networks exists. One of the most used activation

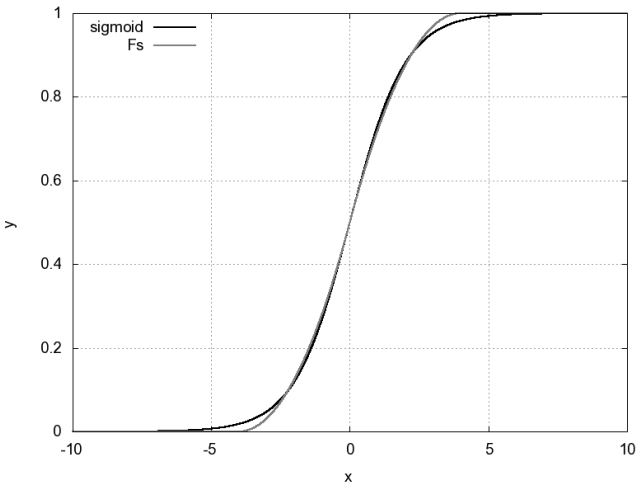


Fig. 3. The sigmoid function and its approximation.

functions in classical neural networks is a *sigmoid* function (3). It is a growing differentiable function, the features of which are important for gradient-descent based learning algorithms like the well known *backpropagation* algorithm [7]. However, this function uses operations of division and power which are not suitable for implementation in hardware. Therefore, it is appropriate to replace it with a more effective approximation with similar features. One of possible approximations is $Fs$ function (7) which is based on equations (4),(5) and (6). Both the sigmoid function graphs and the $Fs$ function are compared in Fig. 3.[8]

$$sigmoid(x) = \frac{1}{1+\epsilon^{-\theta x}} \tag{3}$$

$$\theta = \tfrac{1}{L^2} \quad \beta = \tfrac{2}{L} \tag{4}$$

$$H_s(x) = \begin{cases} x\,(\beta + \theta x) & \text{for } x \in \langle -L, 0\rangle \\ x\,(\beta - \theta x) & \text{for } x \in (0, L\rangle \end{cases} \tag{5}$$

$$G_s(x) = \begin{cases} -1 & \text{for } x \in (-\infty, -L\rangle \\ H_s(x) & \text{for } x \in (-L, L) \\ 1 & \text{for } x \in \langle L, \infty) \end{cases} \tag{6}$$

$$F_s(x) = \frac{1}{2}G_s(x) + \frac{1}{2} \tag{7}$$

## III. Detecting the hard faults

Artificial neural networks are inherently massively parallel structures with a lot of redundancy. Even though this property makes them able to tolerate some faults, this fault tolerance reaches only a certain level and it is complicated to predict its quality. In order to enhance the fault tolerant properties of neural networks, several techniques are used. Some techniques are based on modifications of the neural networks training process to force the networks to learn to be fault tolerant [10], [11], [12]. Other techniques use retraining as a way of recovery from a fault [9], [18]. In some techniques, different modifications and restrictions of weights and neurons activation functions take place [13], [14], [15], [16], [17]. Also, techniques which utilize redundancy are commonly used. Either based on neurons replications [19], [21], [24], [25] or on the well known Triple Modular Redundancy (TMR) technique which is used for both faults detection and masking it in order to produce a correct output.

When implemented in hardware, the neural networks may face two types of faults. The soft (temporary) faults occur only temporarily and affect the computation only for some time. These faults are often caused by radiation generating a Single Event Upset (SEU) - flipping a bit in a memory. This type of fault may be often solved by rewriting the memory with correct data. Other type of faults - hard faults are persistent faults often caused by a physical condition or failure in the device. Most approaches of faults detection, masking and recovery are based on the TMR techniques [22], [23]. This technique, though proven reliable, may fail in some combinations of faults as shown in [20].

We propose a method of permanent faults detection that does not utilize redundancy or learning but it uses properties of neural networks computation instead. It utilizes modifications of basic neural network parameters - weights values and activation function shapes in order to detect a fault using the network output. As we present general principles and algorithm which may be used on any neural network platform and implementation which are flexible enough to allow the needed neural network parameters settings modification, we primarily intent this method to be used on reconfigurable hardware implementations, where a hard fault is more likely to occur and is harder to mask and recover from them in a purely software implementation. In our research, we intent to use these methods, while utilizing a dynamic reconfiguration, with our neural network *Field Programmable Gate Array* (FPGA) implementation. This platform is based on the concept of *Field Programmable Neural Network* (FPNN)[1] and we presented this platform in [4], [5].

### A. Definitions

In order to describe the principles clearly, we declare a set of terms presented in the following list and equations. The terms describe the neural network structure and derive additional terms. Going through the list we define sets and functions containing neurons, synapses and their weights ($N$,$S$ and $W$). Using those, we declare a set of neurons layers $L$, the input layer $I$ and the output layer $O$. For all neurons we define two sets - $\phi$ and $\tau$. The $\phi$ set contains all the sequences of synapses which connect the selected neuron to the input layer (to all its neurons). To the opposite, the $\tau$ set contains all sequences of synapses that connect the neuron to the output neurons.

Based on these sets we define a set $\pi$ for all the neurons which contains all sequences of synapses connecting the input layer to the output layer through the selected neuron. It is important to have this set as we need to find a way to propagate the test data through the network in the sequence that includes

the neuron or synapse we want to test against the presence of a hard fault.

1) $N$ is the set of all neurons.
2) $S \subseteq (N \times N)$ is the set of all synapses.
3) $W = S \to \mathbb{R}$ is the set of weights of all synapses.
4) $W = \{w_s \in \mathbb{R} | s \in S\}$ is the set of weights of all synapses.
5) $B = \{b_n : \mathbb{R}^m \to \mathbb{R} | n \in N \setminus I, m \in \mathbb{N}\}$ is the set of basis functions of all neurons except the input neurons. $m$ is the number of the neuron $n$ inputs.
6) $L$ is set of network layers defined by equation (8).
7) $I$ is the input layer defined by equation (9).
8) $O$ is the output layer defined by equation (10).
9) $F = \{f_n : \mathbb{R} \to \mathbb{R} | n \in N \setminus I\}$ is the set of activation functions of all neurons except the input neurons.
10) $\phi_n$ is a set of sequences of synapses connecting the neuron $n$ to the neurons in the input layer (11).
11) $\tau_n$ is a set of sequences of synapses connecting the neuron $n$ to the neurons in the output layer (12).
12) $\forall n \in N : \pi_n = \phi_n \times \tau_n$ is a set of all sequences of synapses connecting the input layer to the output layer through the neuron $n$.
13) $\psi(s)$ is a sequence of all source neurons of the synapses in the synaptic sequence $s$ (13).
14) $\chi(s)$ is a sequence of all target neurons of the synapses in the synaptic sequence $s$ (14).
15) $\Omega \in \mathbb{R}$ is a chosen global value of weights to be used in further algorithms.
16) $d_a, d_o \in \mathbb{R}$ are chosen input data values for an active neuron ($d_a$) and for other neurons in the input layer ($d_o$).

$$L \subset N^X : \forall (n1, n2) \in S : n1 \in L_1 \wedge n2 \in L_2; \\ L_1, L_2 \in L \tag{8}$$

$$I \in L : \forall n \in I \wedge \exists (n, n_x) \in S : \nexists (n_y, n) \in S; \\ n_x, n_y \in N \tag{9}$$

$$O \in L : \forall n \in O \wedge \exists (n_x, n) \in S : \nexists (n, n_y) \in S; \\ n_x, n_y \in N \tag{10}$$

$$\forall n \in N \exists \phi_n \subset S^X : (n_x, n_{x+1}) \in \phi_n, x \in \{i..m\}; \\ i, m \in \mathbb{N}; n_i \in I; n_{m+1} = n \tag{11}$$

$$\forall n \in N \exists \tau_n \subset S^X : (n_x, n_{x+1}) \in \tau_n, x \in \{m..o\}; \\ m, o \in \mathbb{N}; n_m = n; n_{o+1} \in O \tag{12}$$

$$\forall s = s_1 s_2 .. s_m = (n1, n2)(n2, n3)...(n_{m-1}, n_m) \in S^m : \\ \psi(s) = n1, n2, ..., n_{m-1} \tag{13}$$

$$\forall s = s_1 s_2 .. s_m = (n1, n2)(n2, n3)...(n_{m-1}, n_m) \in S^m : \\ \chi(s) = n2, n3, ..., n_m \tag{14}$$

### *B. Detecting a fault in a synapse without affecting the activation functions*

Using the previous definitions we declare an algorithm which utilizes the modifications of neural networks properties in order to detect the hard fault of a synapse. The principle of the algorithm is to check sequentially all the synapses by propagating the test data to the network and checking the network output while setting all the other synapses weights to 1 in order to omit them from the computation. Omitting the weight ensures that the passing test data are affected only by the weight of the tested synapse which makes the result easy to determine. The algorithm is as follows:

**A) Declare a set $FS = \emptyset$ to store the faulty synapses**.
**B) For all synapses $s \in S$ execute**:

1) To test a synapse $s = (n_1, n_2) \in S$ compute $\pi_{n2}$.
2) Select a sequence $\alpha$ out of $\pi_{n2}$. $\alpha = \beta\gamma, \beta \in \phi_{n2}, \gamma \in \tau_{n2}$.
3) Set all other weights to $\Omega$: $\forall w \in W \setminus \{W(a) | a \in \alpha\} : w = \Omega$.
4) Set weights in $\alpha$ synapses to 1, leave the original value of the tested synapse - $W(s)$: $\forall w \in W(S \setminus \alpha \setminus s) : w = 1$.
5) Present the input data $i$ to the input layer of the network in the form of a vector composed of $d_a$ on the place of the neuron from the $\alpha$ sequence and $d_o$ in places of others input neurons. Let the neural network compute the output $o(\alpha, s, i)$.
6) Compute an $\omega$ value. $\omega(\alpha, s, i) \in \mathbb{R}$ represents the expected output value for the selected synaptic sequence $\alpha$, tested synapse $(n_1, n_2)$ and the input data $i$. It is computed using equation (16) as a sequence of application of all activation functions of the neurons in the $\beta$ sequence over the input data followed by multiplication with the tested synapse weight. Then, the sequence of all activation functions of the neurons in the $\gamma$ sequence is applied.
7) Compute the difference $\epsilon$ between the expected and the actual output value: $\epsilon(\alpha, s, i) = o(\alpha, s, i) - \omega(\alpha, s, i)$.
8) If the difference $\epsilon = 0$ (the output is not affected by a fault), return to the step A). Otherwise execute:
   a) Repeat the steps 1 to 8 for several other $\alpha$ sequences containing the synapse $s$.
   b) If all the $\epsilon$ values are not zero, the synapse $s$ is affected by a fault. Add then the synapse $s$ to the set $FS$.

$$\omega(\alpha, s, i) = f^\gamma \left( f^\beta (i_n) \times w_s \right); \\ n \in N, i_n \in \mathbb{R}, w_s \in S; \\ f^\beta = f_{n1} \circ ... \circ f_{n0}; \psi(\beta) = n0, ..., n1; \\ f^\gamma = f_{n1} \circ ... \circ f_{n0}; \chi(\gamma) = n0, ..., n1 \tag{15}$$

The general problem to deal with in this algorithm is a possibility of neurons outputs saturation preventing the fault detection. This problem has its origin in the input (the $d_a, d_o$ values), used weights values (the $\Omega$ value) during the algorithm

and the activation functions. Regular sigmoid function has the range of $\langle 0, 1\rangle$ and the value of $0.5$ as the function value of zero ($sigmoid(0) = 0.5$). When the weights of synapses outside an $\alpha$ sequence are set to zero, it causes all the neurons in the rest of the network to emit value of $0.5$. This can affect the neurons in the $\alpha$ sequence as well as these values enter their basis functions. Together with data passing through the $\alpha$ sequence this can cause the neuron outputs to be saturated, i.e. to have the value of $1.0$ or $0.0$. When these are valid values for properly functioning network and there are no changes in them, this can cause a fault to be masked from detection. This effect can be straightened even more by values presented to the input neurons both in and outside the $\alpha$ sequence (the $d_a, d_o$ values). It is necessary to choose all these values wisely to obtain as correct detection as possible.

Another problem related to the saturation problem is the problem of false positive detection. In the case that the fault causes the weight to have a high value, it may saturate the successive neuron itself causing saturation of neurons in higher level as well as the saturation of the network outputs. The saturated output then may be detected as fault even in case when other synapses than the faulty one is under the test. This problem may be solved by repetitive detection using different settings as well as using heuristics to obtain specific strategies of the test. This heuristics and methods will be part of the future research.

### C. Detecting a fault in a synapse with affecting the activation functions

The hard faults detection becomes easier and less demanding if there is an option to change shapes of neurons activation functions. By changing the functions to the linear functions $f(x) = x$ we can prevent the saturation problem mentioned in the previous paragraph. However, the saturation problem is still present but it is far less likely as the only risk of saturation is reaching the upper or the lower boundary given by the used data-type and the bit width. Also, by changing the activation functions, we obtain a higher precision of the output computation and lesser influence of neurons faults to the output. The algorithm, derived from the previous algorithm, which utilizes the change of activation functions is as follows:

**A) Declare a set $FS = \emptyset$ to store the faulty synapses**.
**B) For all synapses $s \in S$ perform**:

1) Perform 1) - 4) steps of the section B algorithm.
5) Set the activation functions of the neurons in the $\alpha$ sequence to linear function:
   $\forall n \in \psi(\beta) \cup \chi(\gamma) : f_n(x) = x; f_n(x) \in F$. When the activation function is approximated using the function (7), the modification can be done using constants modifications according to the (17) equation.
6) Perform 5) step of the section B algorithm.
7) Compute an $\omega$ value. $\omega(\alpha, s, i) \in \mathbb{R}$ represents the expected output value for the selected synaptic sequence $\alpha$, tested synapse $(n_1, n_2)$ and the input data $i$. It is computed using equation (16) as a sequence of applications of all activation functions of the neurons in the $\beta$ sequence. In this case, the activation functions are linear, therefore the applications are in principle function of identity. The output data of the activation functions sequence is data followed by multiplication with the tested synapse weight. Then the sequence of all activation functions of the neurons in the $\gamma$ sequence is applied, again as a sequence of identities.
8) Perform 7) - 8) steps of the section B algorithm.

$$\begin{aligned} &\omega(\alpha, s, i) = i_n \times w_s; \\ &n \in N, i_n \in \mathbb{R}, w_s \in S; \end{aligned} \tag{16}$$

### D. Activation functions modifications

If the implementation uses the $Fs$ function as the activation function approximation, it can be simply forced to behave like a linear function in order to pass the neuron input data directly to the output without affecting them by the activation function. In the case of the $Fs$ function, it can be done using constants modifications and input data propagation to the multiplexers realizing the $Gs$ function. The modifications of the functions and the constants are as follows:

$$\begin{aligned} &\theta = 0; \quad \beta = 1 \\ &H_s(x) \quad = \begin{cases} x\,(\beta + \theta x) = x & \text{for } x \in \langle -L, 0\rangle \\ x\,(\beta - \theta x) = x & \text{for } x \in (0, L\rangle \end{cases} \\ &G_s(x) \quad = \begin{cases} x & \text{for } x \in (-\infty, -L\rangle \\ H_s(x) = x & \text{for } x \in (-L, L) \\ x & \text{for } x \in \langle L, \infty) \end{cases} \\ &F_s(x) \quad = 1 \times G_s(x) + 0 = G_s(x) \end{aligned} \tag{17}$$

## IV. Experiments

We have experimented with the first algorithm which does not utilize the activation functions modifications in order to determine the influence of the $d_a, d_o$ and $\Omega$ values to the saturation problem and therefore to the quality of faults detection.

The used neural network was composed of 8 neurons in the input layer, 2 neurons in the output layer and of 64 and 16 neurons in two hidden layers. The sigmoid function was used as the activation function of all the neurons and the function of the weighted sum was used as basis functions. The experiments were implemented using a *FANN library* [3] using 32-bit floating point arithmetic. The neural network was trained to solve the *Diabetes* classification task from the *Proben* [2] set of neural networks benchmark tasks. Every experiment used two identical neural networks, one as a golden model, the second to inject fault and perform the algorithm. Only one fault per test was injected randomly into a synapse and the algorithm was executed to detect the fault.

With each set of $d_a, d_o$ and $\Omega$ values, 100 tests were run and the number of successful detections was measured as a result. The $d_o$ values were chosen in the $\langle -10, 0\rangle$ as we expected that low values around zero may help to prevent the saturation problem as well as the negative values. We expect

these values cause the neurons to emit low values as well which may help to prevent the basis function to generate high values which would saturate the neurons outputs in the higher layers making the neural network output to be saturated as well. The $\Omega$ values were chosen to be the same for the same reasons as it may help to lower the high values emitted by neurons and thus lower the risk of saturation in higher layers. On the other hand, the $d_i$ values are the most important as they enter the computation in the $\alpha$ sequence. In order to explore their influence on the detection quality, we chose them to be in the $\langle -10, 10 \rangle$ interval.

Tables I - III illustrate the results of the experiments. In each table, the $\Omega$ value is the same for all listed experiments and it is declared on the top row of the table. The values of $d_o$ are declared in the third rows of the tables (the first numerical rows) and the $d_i$ values are listed in the first columns of the tables. The cells contain the experiments results illustrating how many detections out of 100 were successful with the $\Omega$, $d_i$ and $d_o$ set according to the position in the table.

TABLE I
THE EXPERIMENTS RESULTS WHEN $\Omega = 0.0$

| $\Omega = 0.0$ | | | | | |
|---|---|---|---|---|---|
| | $d_o$ | | | | |
| $d_i$ | **-10.0** | **-1.0** | **-0.1** | **-0.01** | **0.0** |
| **-10.0** | 21 | 21 | 56 | 21 | 24 |
| **-1.0** | 20 | 99 | 32 | 58 | 78 |
| **-0.1** | 66 | 99 | 72 | 1 | 97 |
| **-0.01** | 81 | 100 | 0 | 0 | 71 |
| **0.01** | 70 | 96 | 79 | 58 | 0 |
| **0.1** | 79 | 100 | 75 | 63 | 85 |
| **1.0** | 86 | 100 | 83 | 100 | 82 |
| **10.0** | 100 | 100 | 1 | 63 | 91 |

TABLE II
THE EXPERIMENTS RESULTS WHEN $\Omega = -0.01$

| $\Omega = -0.01$ | | | | | |
|---|---|---|---|---|---|
| | $d_o$ | | | | |
| $d_i$ | **-10.0** | **-1.0** | **-0.1** | **-0.01** | **0.0** |
| **-10.0** | 97 | 97 | 98 | 92 | 75 |
| **-1.0** | 98 | 98 | 98 | 100 | 81 |
| **-0.1** | 99 | 99 | 100 | 99 | 70 |
| **-0.01** | 100 | 96 | 97 | 96 | 64 |
| **0.01** | 96 | 98 | 100 | 99 | 61 |
| **0.1** | 100 | 100 | 100 | 98 | 68 |
| **1.0** | 99 | 100 | 100 | 97 | 72 |
| **10.0** | 100 | 100 | 100 | 96 | 80 |

As you can see in the tables, the most of the experiments resulted in the high ratio of detected faults. If we are going to identify the general trends of the $\Omega$, $d_i$ and $d_o$ values influence on the results, we can see that higher values of $d_i$ often led to better results. This can be also stated in general about influence of the higher negative values of $d_o$. As we expected, the negative values of $d_o$ helped the detection by lowing the risk of saturation and allowing the fault detection by doing so. On the other hand, the $0.0$ value of $d_o$ proved to provide generally worse results. As it was said before, the value of $0.0$ as an input to the neuron causes to emit the value of $0.5$ as a result in case it uses the sigmoid function as an activation function, which is the case of these experiments. Also, the high negative values of $d_i$ provided worse results as they probably increased the saturation problem in case the weights values were high in the $\alpha$ sequence or the injected fault had a high value.

TABLE III
THE EXPERIMENTS RESULTS WHEN $\Omega = -10.0$

| $\Omega = -10.0$ | | | | | |
|---|---|---|---|---|---|
| | $d_o$ | | | | |
| $d_i$ | **-10.0** | **-1.0** | **-0.1** | **-0.01** | **0.0** |
| **-10.0** | 75 | 55 | 53 | 54 | 60 |
| **-1.0** | 99 | 84 | 84 | 88 | 64 |
| **-0.1** | 84 | 91 | 92 | 100 | 58 |
| **-0.01** | 88 | 85 | 93 | 87 | 43 |
| **0.01** | 74 | 85 | 92 | 95 | 64 |
| **0.1** | 81 | 89 | 94 | 99 | 60 |
| **1.0** | 81 | 86 | 89 | 89 | 55 |
| **10.0** | 100 | 89 | 100 | 98 | 78 |

TABLE IV
THE EXPERIMENTS RESULTS WHEN $\Omega = -0.1$

| $\Omega = -0.1$ | | | | | |
|---|---|---|---|---|---|
| | $d_o$ | | | | |
| $d_i$ | **-10.0** | **-1.0** | **-0.1** | **-0.01** | **0.0** |
| **-10.0** | 97 | 97 | 98 | 93 | 82 |
| **-1.0** | 99 | 98 | 99 | 98 | 85 |
| **-0.1** | 99 | 99 | 100 | 100 | 73 |
| **-0.01** | 99 | 98 | 98 | 99 | 61 |
| **0.01** | 96 | 100 | 100 | 100 | 64 |
| **0.1** | 100 | 100 | 100 | 99 | 77 |
| **1.0** | 99 | 100 | 99 | 100 | 83 |
| **10.0** | 100 | 100 | 100 | 97 | 77 |

TABLE V
THE EXPERIMENTS RESULTS WHEN $\Omega = -1.0$

| $\Omega = -1.0$ | | | | | |
|---|---|---|---|---|---|
| | $d_o$ | | | | |
| $d_i$ | **-10.0** | **-1.0** | **-0.1** | **-0.01** | **0.0** |
| **-10.0** | 21 | 24 | 56 | 21 | 24 |
| **-1.0** | 20 | 58 | 32 | 58 | 66 |
| **-0.1** | 66 | 97 | 72 | 1 | 97 |
| **-0.01** | 81 | 71 | 0 | 0 | 71 |
| **0.01** | 70 | 91 | 79 | 58 | 0 |
| **0.1** | 79 | 85 | 75 | 63 | 85 |
| **1.0** | 86 | 100 | 83 | 100 | 100 |
| **10.0** | 100 | 91 | 1 | 63 | 91 |

As the tables II and IV illustrate, the low negative values of $\Omega$ has positive influence on the results. These values reduced the neurons output to higher layers helping to prevent the saturation problem. The assumption of false positive detection during the experiments was also confirmed, however this aspect of the problem is beyond the range of this paper.

## V. CONCLUSIONS AND FUTURE RESEARCH

In this paper, we described basis of neural networks as well of formal basis of two algorithms which are the core of this research. The algorithms offer the way to detect hard fault in neural network synapses. Both algorithms are based on the

principle of setting the synapses weights to some chosen value, then creating an interconnected sequence from input layer to an output layer. One of the synapses is then set with its original weight and chosen testing data are passed trough the network and the output is compared to the pre-calculated valid result. The difference between the outputs indicates a fault.

One of the algorithms uses the change of activation functions to linear ones preventing the problem with an output saturation which may occur with the other algorithm. The experimental part of this work focuses on this problem and shows the effect of the chosen values of the input data and the weights to the quality of faults detection. The results show that combination of low negative $\Omega$ values with high values of $d_o$ and negative values of $d_o$ led in general to the best results as they were the most successful preventing the saturation of the network output.

In the future research, more extensive experiments with both algorithms will be done as well as an optimization of the algorithms based on a test strategy selection heuristics. The heuristics are needed to achieve higher speed and better precision of detection as well as saving of resources. Also, the heuristics will help to prevent false positive detection which may occur in case of the algorithm which does not utilize the activation function modification. In addition, experiments with limited precision will be performed, as in case of classical neural networks, 16-bit fixed point precision was proven to be sufficient [6]. We have also designed modifications for both algorithm to be used to detect fault in both neurons basis functions and their activation functions.

## ACKNOWLEDGMENT

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science – LQ1602, the BUT project FIT-S-17-3994 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

## REFERENCES

[1] Girau, B.: FPNA: Concepts and Properties. In *FPGA Implementations of Neural Networks*, A. R. Omondi; J. C. Rajapakse, Springer US, 2006, ISBN 978-0-387-28487-3, pp. 71–123, http://dx.doi.org/10.1007/0-387-28487-7-3

[2] Prechelt, L. P.; Informatik, F. F.: — A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical report, Universitat Karlsruhe; 76128 Karlsruhe, Germany, 1994.

[3] Fast Artificial Neural Network Library (FANN). http://leenissen.dk/fann/wp/

[4] KRCMA Martin, KASTIL Jan a KOTASEK Zdenek: *Mapping trained neural networks to FPNNs.* In: IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems. Belgrade: IEEE Computer Society, 2015, pp. 157–160. ISBN 978-1-4799-6779-7.

[5] Krcma, M.; Kotasek, Z.; Kastil, J.: Fault tolerant Field Programmable Neural Networks. In *Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC), 2015*, Oct 2015, pp. 1–4, 10.1109/NORCHIP.2015.7364381.

[6] Holt, J.; Baker, T.: Back propagation simulations using limited precision calculations. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume II, July 1991, pp. 121 –126.

[7] Munakata, T.: Neural Networks: Fundamentals and the Backpropagation Model. In *Fundamentals of the New Artificial Intelligence*, editace T. Munakata, Texts in Computer Science, Springer London, 2007, ISBN 978-1-84628-839-5, pp. 7–36, http://dx.doi.org/10.1007/978-1-84628-839-5--2

[8] Kwan, H.: Simple sigmoid-like activation function suitable for digital hardware implementation. *Electronics Letters*, 1992: pp. 1379–1380. http://link.aip.org/link/?ELL/28/1379/1

[9] Deng, J.; Rang, Y.; Du, Z.; aj.: Retraining-based timing error mitigation for hardware neural networks. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, March 2015, s. 593–596.

[10] Elsimary, H.; Mashali, S.; Shaheen, S.: Generalization ability of fault tolerant feedforward neural nets. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, Issue 1, Oct 1995, pp. 30–34 vol.1, 10.1109/ICSMC.1995.537728.

[11] Arad, B. S.; El-Amawy, A.: On Fault Tolerant Training of Feedforward Neural Networks. *Neural Networks*, Issue 10, vol. 3, 1997: pp. 539 – 553, ISSN 0893-6080, http://dx.doi.org/10.1016/S0893-6080(96)00089-5. http://www.sciencedirect.com/science/article/pii/S0893608096000895

[12] Ito, T.; Takanami, I.: On fault injection approaches for fault tolerance of feedforward neural networks. In *Test Symposium, 1997. (ATS '97) Proceedings., Sixth Asian*, Nov 1997, ISSN 1081-7735, pp. 88–93, 10.1109/ATS.1997.643927.

[13] Haruhiko, T.; Hidehiko, K.; Terumine, H.: Partially weight minimization approach for fault tolerant multilayer neural networks. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, vol. 2, 2002, ISSN 1098-7576, pp. 1092–1096, 10.1109/IJCNN.2002.1007646.

[14] Haruhiko, T.; Hidehiko, K.; Terumine, H.: Fault tolerant training algorithm for multi-layer neural networks focused on hidden unit activities. In *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, 2006, pp. 1540–1545, 10.1109/IJCNN.2006.246616.

[15] Hammadi, N. C.; Ito, H.: A Learning Algorithm for Fault Tolerant Feedforward Neural Networks. *IEICE Trans. Information and Systems*, Issue 80, 1996: pp. 21–27.

[16] Rusiecki, A.: Fault tolerant feedforward neural network with median neuron input function. *Electronics Letters*, Issue 41, vol. 10, May 2005: pp. 603–605, ISSN 0013-5194, 10.1049/el:20058169.

[17] Kamiura, N.; Taniguchi, Y.; Isokawa, T.; and col.: An improvement in weight-fault tolerance of feedforward neural networks. In *Test Symposium, 2001. Proceedings. 10th Asian*, 2001, ISSN 1081-7735, pp. 359–364, 10.1109/ATS.2001.990309.

[18] Sequin, C.; Clay, R.: Fault tolerance in artificial neural networks. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, June 1990, pp. 703–708 vol.1, 10.1109/IJCNN.1990.137651.

[19] Phatak, D.; Koren, I.: Complete and partial fault tolerance of feedforward neural nets. *Neural Networks, IEEE Transactions on*, Issue 6, vol. 2, Mar 1995: pp. 446–456, ISSN 1045-9227, 10.1109/72.363479.

[20] Tohma, Y.; Koyanagi, Y.: Fault-tolerant design of neural networks for solving optimization problems. *Computers, IEEE Transactions on*, Issue 45, vol. 12, Dec 1996: pp. 1450–1455, ISSN 0018-9340, 10.1109/12.545976.

[21] Zhou, Z.-H.; Chen, S.-F.; Chen, Z.-Q.: Improving tolerance of neural networks against multi-node open fault. In *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, Issue 3, 2001, ISSN 1098-7576, pp. 1687–1692 vol.3, 10.1109/IJCNN.2001.938415.

[22] Mahdiani, H. R.; Fakhraie, S. M.; Lucas, C.: Relaxed Fault-Tolerant Hardware Implementation of Neural Networks in the Presence of Multiple Transient Errors. *IEEE Transactions on Neural Networks and Learning Systems*, Issue 23, vol. 8, Aug 2012: pp. 1215–1228, ISSN 2162-237X, 10.1109/TNNLS.2012.2199517.

[23] Latif-Shabgahi, G.; Hirst, A.; Bennett, S.: A novel family of weighted average voters for fault-tolerant computer control systems. In *European Control Conference (ECC), 2003*, Sept 2003, pp. 642–646.

[24] Emmerson, M.; Damper, R.: Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application. *Neural Networks, IEEE Transactions on*, Issue 4, vol. 5, Sep 1993: pp. 788–793, ISSN 1045-9227, 10.1109/72.248456.

[25] Ahmadi, A.; Sargolzaie, M. H.; Fakhraie, S. M.; aj.: A Low-Cost Fault-Tolerant Approach for Hardware Implementation of Artificial Neural Networks. In *Computer Engineering and Technology, 2009. ICCET '09. International Conference on*, Issue 2, Jan 2009, pp. 93–97, 10.1109/ICCET.2009.204.