# Error Resilient Neuromorphic Networks Using Checker Neurons

*(Invited Paper)*

Sujay Pandey, Suvadeep Banerjee and Abhijit Chatterjee,
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, USA

***Abstract*—The last decade has seen tremendous advances in the application of artificial neural networks to solving problems that mimic human intelligence. Many of these systems are implemented using traditional digital compute engines where errors can occur during memory accesses or during numerical computation. While such networks are inherently error resilient, specific errors can result in incorrect decisions. This work develops a low overhead error detection and correction approach for multilayer artificial neural networks, here the hidden layer functions are approximated using *checker neurons*. Experimental results show that a high coverage of injected errors can be achieved with extremely low computational overhead using *consistency properties of the encoded checks*. A key side benefit is that the checks can flag errors when the network is presented outlier data that do not correspond to data with which the network is trained to operate.**

## I. Introduction

There has been significant interest in non-Von Neumann architectures for computing in the recent past due to the projected end of Moore's law of scaling [1]. Machine learning algorithms based on neural network architectures [2], [3] have emerged as an exciting new paradigm for information technology with applications in artificial intelligence, image/signal processing and control. This has spurred interest in implementations of such architectures in VLSI using conventional multi-processors [4] and semi-synchronous machines such as IBM's True North chip [5] based on spiking neural networks. A key problem that emerges in the silicon implementation of neuromorphic systems is the tolerance of such systems to soft errors induced by ultra low voltage operation and environmental noise. While neuromorphic systems have some degree of resilience to data input and computational errors [6], there is always fear that when these errors become large (such as when errors occur in the most significant bits of arithmetic computation), the numerical results or classification decisions made by such systems will be incorrect. There has been work in the past on designing failure-tolerant neuromorphic systems [7], [8], [9]. However, these target either permanent failure modes or incur significant overhead for implementing failure tolerance. In contrast, this paper develops a low cost soft-error error resilience approach for multilayer perceptron networks. In this approach, one or more *checker neurons* are used to approximate the functions of a set of neurons in the input, output or hidden layers of a multilayer perceptron network. Typically, one checker neuron is used to check the functions of five or more neurons of the neural network using a *consistency checking algorithm.*. A Checker Neuron is placed between two hidden layers such that it takes input from the previous layer and is trained to match the summation of outputs of the next layer. The goal is to design the checking circuitry in such a way that the overhead of error resilience is less than 20%. It is seen that high (detection) coverage of soft errors that actually affect the results produced by the neural network is achieved across diverse test cases.

The fundamental contributions of this work are:

- Real time error detection is demonstrated using encoded consistency checks for artificial neural networks using low computational overhead.
- An approximate error correction approach is proposed which produces close to the target output results from the neural network. This method is mainly targeted towards deep neural network applications.
- It is shown that through the use of the proposed consistency checks, the presence of outlier input data(as defined earlier) is detected in real time without the need to do extensive statistical analysis of network input data.

The rest of the paper is organized as follows. Section II provides background related to the proposed work. Section III describes the proposed approach. Section IV presents experimental results. Conclusions and future work are discussed in Section V.

## II. Background

A multilayer perceptron also called as an Artificial Neural Network consists of an input layer, output layer and one or more hidden layers in between them. The hidden layers consist of neurons which are the weighted summations of outputs of the previous layer with non-linear activation functions. Figure 1 shows a general neural network with inputs $I_1$ to $I_n$, outputs $O_1$ to $O_m$ and N hidden layers with p neurons each. The bias vector $B_i$ is defined as $\bar{B_i} = [b_{i,1}b_{i,2}...b_{i,p}]$ The output of the $j_{th}$ neuron in the $i_{th}$ layer $HL_{i,j} = f(WS_{i,j} + b_{i,j})$ where $WS_{i,j} = \sum_{k=1}^{p} W_{i-1,j}^{i,j} * HL_{i-1,k}$. Here $W_{i-1,k}^{i,j}$ is the weight by which the output of the $k_{th}$ neuron is multiplied before being input to the activation function f() of the $j_{th}$ neuron in layer i. This is easily generalized to networks with different number of neurons in all hidden layers. The non linear function f,considered in this work, also called as activation function is a sigmoid function defined as $f(x) = \frac{1}{(1+e^{-x})}$. Other activation functions can be hyperbolic tangent, hard limiters etc. Figure 2

978-1-5386-5992-2/18/$31.00 

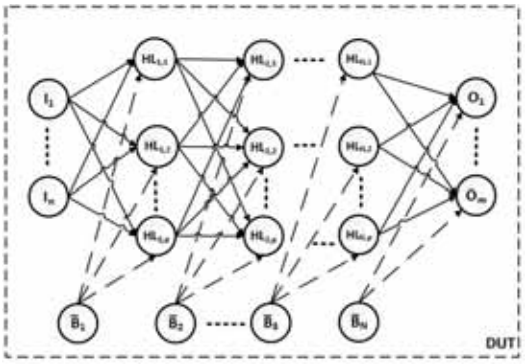


Fig. 1. A Simple Artificial Neural Network

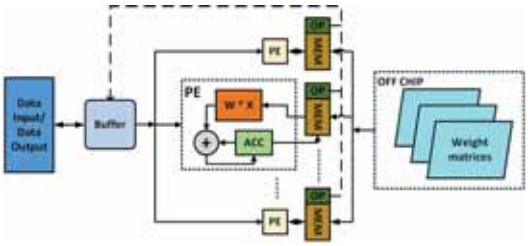


Fig. 2. Hardware Implementation of a Artificial Neural Network

shows the most common implementation of general neural network computations. A neuron is represented by a processing element (PE) which consists of a multiply and accumulate unit. The weight matrices are stored in off-chip memory as shown. These weights are fetched into the local memory (MEM). The PE performs the weighted summation and finally the activation function is applied and the data output (OP) is fed to buffer. In this kind of implementation errors can generally occur during the calculation of the weighted sum or the activation function. Some errors in the neural network might not affect the output significantly whereas can cause a significant error. These critical errors can make the overall system unreliable. The proposed work deals with soft errors induced by radiation which result in the form of bit or word errors.

## III. Proposed Approach

In this work the engine data-set [10] which has Fuel rate and Speed as the two inputs to the system and Torque and Nitrous oxide emissions as the two outputs of the system is considered. It consists of 1199 samples. The network considered here consists of two hidden layers with three neurons per hidden layer.

### A. Error detection using consistency checks

Consistency checks performed by introducing extra neurons between the layers of neural network. These extra neurons called the *checker neurons* approximate the functions of hidden and input/output layer neurons with lower order non linear approximation. As shown in Figure 3, the checker network with an additional bias is deployed between two hidden layers. There may be one or more checker neurons in the checker network (only one checker neuron is shown in Figure 3 for ease of explanation). The checker neurons are *trained after*

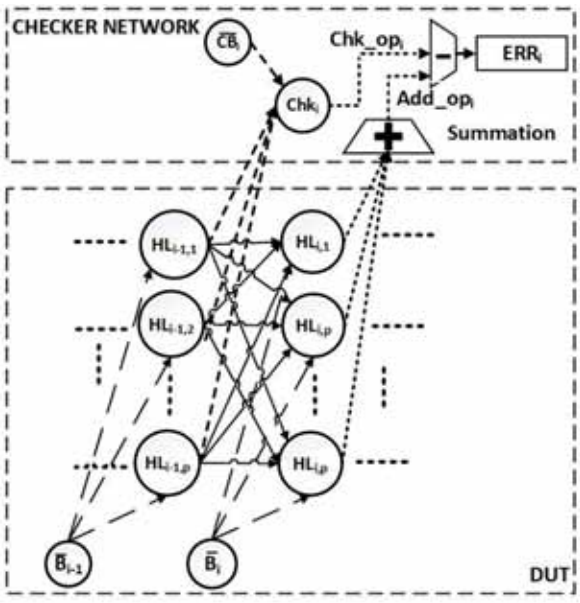


Fig. 3. Artificial Neural Network with Checker Module

*the original network has been trained* to correlate with the summation of the next layer of the trained network(DUT). The output of the checker neuron is calculated as follows.

$$WSchk_{i,j} = \sum_{k=1}^{p} V_{i-1,k}^{i,j} * HL_{i-1,j} \quad (1)$$

$$Chk\_op_i = f(WSchk_{i,j} + b_{i,j}) \quad (2)$$

$V_{i-1,j}$ are the weights of the checker network, $WSchk_{i,j}$ is the weighted sum at the checker neuron, $HL_{i-1,j}$ are the outputs of previous layer, $C\bar{B}_i$ is the bias of the checker neuron $Chk_i$ and $Chk\_op_i$ is the checker output. The output

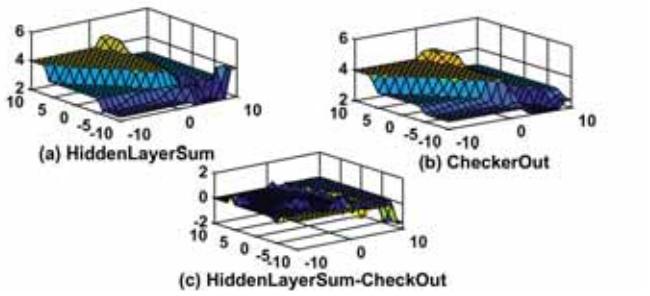


Fig. 4. Correlation between checker Neuron Vs Hidden layer summation

of the adder block is $Add\_op_i = \sum_{j=1}^{p} HL_{i,j}$ The overall error of the system as shown in Figure 3 is computed as: $ERROR_i = Add\_op_i - Chk\_op_i$ The error value, $ERR_i$, is expected to stay within a certain threshold and go out of this threshold if an error occurs in the system. To explain this a simple neural network with two inputs, two outputs and two hidden layers with five neurons is presented. Here the weights and biases of the system were randomly chosen

and a uniformly sampled input stimulus between -10 to +10 was applied to the system. The checker neuron was added as explained earlier and was trained to correlate with the sum of the next layer. Figure 4(a) output of the summation of next hidden layer neurons(HiddenLayerSum),Figure 4(b) shows the output from checker neuron (CheckerOut) and Figure 4(c) shows the difference between them. From this figure it is apparent that the correlation between the checker neuron output and the hidden layer summation is significant.

### B. Error correction approach

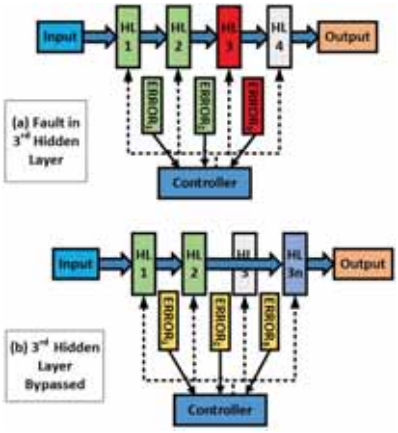


Fig. 5. Error correction scheme

When an error in the $i^{th}$ layer is flagged (ERROR in Figure 5 is non zero), the computation of the respective hidden layer is bypassed. However, since there is one less functional layer of neurons in the network than before, the weights and biases of neurons associated with all layers that are fed by the error affected layer, need to be recalculated. The objective is to make sure the overall neural network produces close to target results using computations from one less layer than the original network. In each error case corresponding errors in the $i^{th}$ layer of the neural network, the weights and biases of all successive layers that result in acceptable network computation are precomputed using the same training data set used to train the original network. These precomputed weights and biases are stored in look up table that can be dynamically updated in the neural network architecture as shown in Figure 2 without any loss of computation throughput by merely indexing a different array in memory.

The system shown in the Figure 5 consists of checkers between network layers (for brevity checker is only shown for the hidden layers) called ERROR and a controller. The third hidden layer shown in the figure called HL3 has an error and as a result $ERROR_3$ sends an error flag to the controller. Since the system was already trained for a three hidden layer Network and the output till Hidden Layer 2 is not erroneous, the controller bypasses the 3rd Hidden layer and updates the weights for the 4th Hidden layer for a three Hidden layer Network. Once the error is corrected the system comes back to the four layer system until the next error occurs. The same concept will apply to the other layers of the system.

## IV. Experimental Results

For the experiments in this work, the soft error model chosen is a single bit-flip model where the error injected module is chosen randomly (from the activation function computations and the weighted-sum calculations) and a randomly selected bit of the word representation of corresponding data is flipped for a random input data to the neural network.

### A. Experiment 1: Nonlinear Function Approximation

As the first test case, we choose the appropriate neural networks for nonlinear function approximations in a few data-sets and demonstrate error detection and correction results.

*1) Error detection using Consistency Checks:* In this experiment, the same neural network with two hidden layers trained with the engine data-set is selected. Errors are injected both in the weighted sum and the activation function computations randomly. The system consists of two inputs and two outputs. In an error-free operation of the network, the predicted outputs of the trained network is slightly different from the target outputs due to training error. The range of differences of both outputs O1 and O2 from the target outputs are computed for the training set and the maximum absolute difference in either outputs is defined as $\Delta$O1 and $\Delta$O2. Now, each injected error in the system operation is classified as a '*blue*' or '*red*' error according to this notion - if the injected error corrupts either one of the outputs from their corresponding target outputs more than $\Delta$O1 or $\Delta$O2 respectively, it is a red error, whereas if the prediction error of both outputs are less than $\Delta$O1 or $\Delta$O2, it is a blue error. 1000 different errors are randomly injected while the neural network is provided with the entire 1199 data samples. 1 checker neuron is implemented for the purpose of consistency checking, The distribution of red and blue errors on the checker module output value is shown in Figure 6.

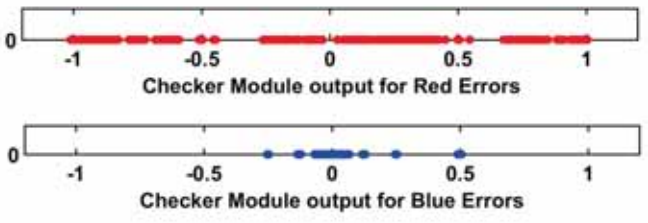


Fig. 6. Error detection for Engine data-set

As shown in Figure 6, the red and blue errors are plotted on the checker module output value. The blue errors are limited to a small value while the red errors are larger in value. This shows that the checker module output indicates the criticality of injected errors to the network's operation. If the detection threshold of the checker module is decided based on Figure 6, a few red errors will be undetected and be classified as false negatives. However, it is observed that these red errors (false negatives) are less critical than

the red errors lying outside the threshold chosen from the boundary of blue errors. Fault coverage is calculated as the percentage of red errors found outside the threshold from the total red errors. The experiment was conducted for different

TABLE I
TEST COVERAGE FOR DIFFERENT TEST CASES

| Test Case | Coverage | Checker neurons | Hidden Layer Neurons |
|---|---|---|---|
| Engine Data set | 81.90% | 2 | 5 |
| Chemical Data set | 85.32% | 1 | 3 |
| Cholesterol Data set | 92.67% | 1 | 3 |
| Body-fat Data set | 83.86% | 1 | 3 |

data-sets(Chemical dataset, Cholesterol dataset and Bodyfat dataset) from MATLAB[10]. Table 1 shows the Fault coverage for different data-sets with different hidden layer neurons and the checker neurons.

### B. Experiment 2: Character recognition problem

Here a Neural Network with two hidden layers is trained for character recognition. The training set consists of ten alphabets


Fig. 7. Training set for character recognition test case

represented in a 9X7 image with the pixel values between 0 and 1.

The single numeric output of the system is interpreted as the corresponding alphabet, for example output is '1' for 'A' etc. . The test set is created with the random noise injection into the training set. The four training and test set characters are shown in Figure 7. Outlier detection experiment is performed for this test case with the help of checker module:

*1) Input data set outlier detection:* An important problem in neural network research is that of determining *in real time* whether the network is able to process the input data provided to it in a reliable manner. *The neural network output cannot be trusted when it is presented input data that it is not trained to respond to.* This is called *input data set outlier detection*. It is seen that the checking mechanism of Figure 4 is extremely effective in such outlier detection. This is because the data values produced by $Chk\_op_i$ and $Add\_op_i$ of Figure 4 are designed to be identical across the data set that the neural network is trained for. However, the probability that the same holds when the neural network is presented outlier data is very low. Currently, outlier input data detection is possible only through extensive statistical analysis and is hard to perform in real time. In the test case described earlier for character recognition, characters from the outlier set 7,N,U,X,O and W were misclassified as G,J,F,B,I and D respectively by the neural network.

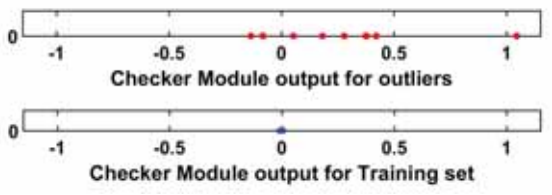


Fig. 8. Outlier detection for character recognition test case

In simulation experiments, red dots and blue dots are used represent ERROR values produced when the neural network is presented with outlier data and training data respectively. Figure 8 shows ERROR values for the character recognition problem when known (unknown) characters are presented to the neural network for analysis. *It is seen that 100% accuracy of input data set outlier detection is achieved in real-time.*

## V. CONCLUSIONS

In this work, a novel scheme to make Neural Networks error resilient was proposed. Consistency Checks is proposed for error detection. With a minimal overhead a substantial fault coverage was observed as explained in the experimental results. Another significant contribution is the outlier detection result presented using the consistency checks.


## ACKNOWLEDGMENT

This research was supported in part by NSF Grant S&AS: 1723997 and by a matching grant from the Center for Co-Design of Chip, Package and System at Georgia Tech.


## REFERENCES


[1] T.Theis and H. S. P. Wong, "The end of moore's law: A new beginning for information technology," in *Computing in Science and Engineering*, vol. 19, no. 2, 2017, pp. 41–50.

[2] J. Hertz, J. , K. , A. Flisberg, P. , and R. G, *Introduction To The Theory Of Neural Computation*, 01 1991, vol. 44.

[3] R. J. Schalkoff, *Artificial neural networks /.* New York :: McGraw-Hill, 1997.

[4] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *Proceedings, Design Automation and Test in Europe*, March 2015, pp. 701–706.

[5] F. Akopyan, et. al., and D. Modha, "Truenorth: Design and tool flow of a 65mw, 1 million neuron programmable neurosynaptic chip," in *IEEE Transactions on Computer-Aided Design*, vol. 34, no. 10, October 2015, pp. 1537–1557.

[6] S. Pandey, S. Banerjee, and A. Chatterjee, "Concurrent error detection and tolerance in kalman filters using encoded state and statistical covariance checks," in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, July 2016, pp. 161–166.

[7] 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, Mar 1995.

[8] E. Sugawara, M. Fukushi, and S. Horiguchi, "Fault tolerant multi-layer neural networks with ga training," in *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, Nov 2003, pp. 328–335.

[9] G. Jian and Y. Mengfei, "Evolutionary fault tolerance method based on virtual reconfigurable circuit with neural network architecture," *IEEE Transactions on Evolutionary Computation*, vol. PP, no. 99, pp. 1–1, 2017.

[10] "Matlab dataset." [Online]. Available: https://www.mathworks.com/help/nnet/gs/neural-network-toolbox-sample-data-sets.html