ReiNN: Efficient Error Resilience in Artificial Neural Networks using Encoded Consistency Checks

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

***Abstract*—In this research, a low cost error detection and correction approach is developed for multilayer perceptron networks, where *checker neurons* are used to encode hidden layer functions using independent training experiments. Error detection and correction is predicated on validating *consistency properties of the encoded checks* and shows that high coverage of injected errors can be achieved with extremely low computational overhead.**

I. Introduction

In the recent past, machine learning algorithms based on neural network architectures [2]–[4] 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 [5] and semi-synchronous machines such as IBM's True North chip [6] based on spiking neural networks. While neuromorphic systems have some degree of resilience to data input and computational errors, 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]–[10]. 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* per input, output or hidden layer of the original neural network are trained to compute a linear function of the outputs of all the neurons in the layer of the neural network being checked. Errors are detected by checking for differences between the value generated by the checker neurons vs. the linear function of the set of neurons being checked in the neural network layer (called a *consistency check*). A methodology for real-time error compensation is proposed. This produces results that are *close* to expected results. The fundamental contributions of this work are:

- This work demonstrates, for the first time, real time error detection using encoded consistency checks for artificial neural networks using minimal hardware overhead. More specifically, we propose addition of extra neuron(s) between the hidden layers and input/output of the artificial neural networks. Errors that occur during multiplications and additions as well as computation of the neural network activation functions are detected [11].
- For critical neural network applications, an approximate error correction approach is developed that delivers close to expected output results from the network.

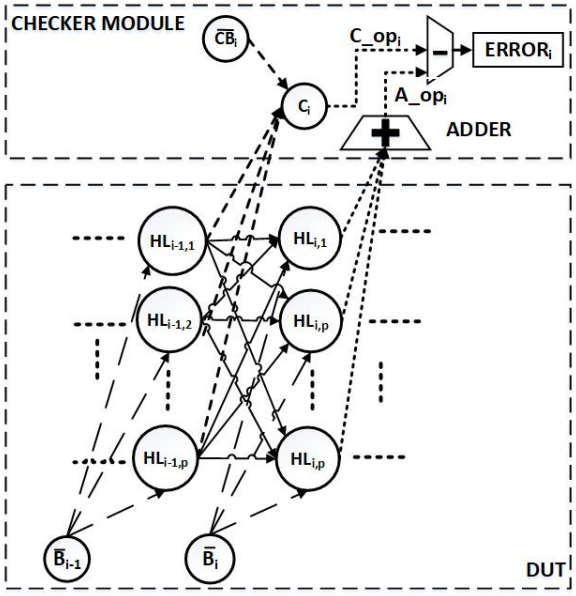

Fig. 1. Artificial Neural Network with Checker Module

II. Proposed Approach

Here we have considered the engine data-set [12] 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. The data-set consists of 1199 samples. Here the neural network is trained to estimate the engine torque and emissions from its fuel use and speed. We consider a two hidden layer neural network with three neurons per hidden layer.

A. Error detection approach using Consistency checks

Error detection is performed by introducing checker neurons into the layers of the neural network. These neurons approximate the functions of hidden and input/output layer neurons with lower order non linear approximations. As shown in Figure 1, the checker network is deployed between two

978-1-5386-3728-9/18/$31.00 ©2018 IEEE

hidden layers with an additional bias. The checker network may contain one or more checker neurons (only one checker neuron is shown in Figure 1 for ease of explanation). The checker neurons are *trained* to produce a linear sum of the outputs of the neurons in the layer being checked. The output of the checker neuron is calculated as follows.

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

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

$V_{i-1,j}$ 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 C_i and C_op_i is the checker output.

The output of the adder is $A\_op_i = \sum_{j=1}^{p} HL_{i,j}$ The overall error of the system as shown in Figure 1 is computed as: $ERROR_i = A\_op_i - C\_op_i$ The error value, $ERROR_i$, lies within calibrated threshold values under nominal network operation and exceeds this threshold under error.

B. Error correction approach

When an error in the i^{th} layer is flagged, the computation of the respective hidden layer is bypassed. However, since the network has one less functional layer of neurons than before, the weights and biases of neurons associated with all layers that are fed by the error affected layer, need to be recomputed. This is done so that the overall neural network produces acceptable results using computations from one less layer than the original network. For each error scenario corresponding errors in the i^{th} layer of the 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 weights are stored in a look up table that is dynamically updated in the neural network without any loss of computation throughput by merely indexing a different array in memory. This approach is useful only in the cases where the neural network consists of many hidden layers.

C. Error Coverage

For our experiments, the soft error model chosen is a single bit flip model where the error injected module is chosen randomly (among the activation function computations or the weighted-sum calculations) and a randomly selected bit of the word representation of corresponding data is flipped for randomly selected input data to the network. For the engine data set, the network consists of two inputs and two outputs, 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 is calculated as ΔO1 and ΔO2. An injected error in the system is considered critical and included in coverage calculations only if it causes the output to deviate beyond ΔO1 or ΔO2. Error coverage data for 1000 randomly injected errors for different data sets is shown in Table I.

TABLE I
ERROR COVERAGE USING CONSISTENCY CHECKS

Data-sets	**Engine**	**Chemical**	**Cholesterol**	**Body-fat**
Hidden Layer Neurons	5	3	4	4
Checker Neurons	2	1	1	1
Error coverage	81.90%	85.32%	73.75%	68.84%

III. CONCLUSIONS AND FUTURE WORK

In this work, a novel scheme for error resilience in neural networks is proposed. Consistency checks were used for error detection and an approximate error correction methodology was developed. Results show that high coverage of critical soft errors is achieved with less than 20 percent computational overhead.

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

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

[3] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1994.

[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] 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.

[7] 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.

[8] A. P. Johnson, J. Liu, A. G. Millard, S. Karim, A. M. Tyrrell, J. Harkin, J. Timmis, L. J. McDaid, and D. M. Halliday, "Homeostatic fault tolerance in spiking neural networks: A dynamic hardware perspective," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–13, 2017.

[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] 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.

[11] M. T. Hagan, "Engine dataset." [Online]. Available: http://hagan.okstate.edu