Exploiting Hidden Layer Modular Redundancy for Fault-Tolerance in Neural Network Accelerators

Schuyler Eldridge, Ajay Joshi
Boston University

Abstract

Neural network accelerators are an increasingly utilized component of heterogeneous multicore architectures. This new utilization stems from their capability to improve the power–performance of machine learning algorithms and emerging techniques like core state prediction and code approximation. We explore the exploitation of the inherently redundant structure of neural networks for fault-tolerance. While fault-tolerance in neural networks is far from guaranteed or trivial to extract, we present preliminary work that uses a basic, modular redundant approach to provide an improved level of fault-tolerance for applications using neural networks.

1. Introduction

As CMOS technology scales to smaller dimensions, computer architects have embraced novel designs to continue to reduce power and improve performance. Presently, multicore architectures are ubiquitous, heterogeneous architectures are common, and dedicated computational accelerators for general- or special-purpose applications are becoming widespread. One type of accelerator architectures that is being actively investigated is that of neural network accelerators [7].

Neural networks (NNs) are a subset of machine learning approaches that mimic the functionality and connective structure of biological neurons and can be used for classification and regression applications. Biological systems demonstrate significant fault-tolerance and resilience to damage due to their structure, defense (immune system), and repair mechanisms. *Artificial* NNs, being an approximate model of parts of biological systems, are not, as may naively be expected, fault-tolerant [5]. Nevertheless, clever techniques have been developed that enable the redundant structure of NNs to be leveraged to improve NN fault-tolerance. These approaches stem from a desire to create a *balanced* NN where the computational importance of connections in the NN are approximately equal. Such techniques include exposing the NN to the types of expected faults during training [6, 2, 1, 5] as well as setting bounds on NN weights and pruning NNs to remove unimportant neurons or split important ones [1]. Additionally, simple retraining [6] of a faulty NN or introducing modular redundancy [4] can achieve improved fault-tolerance.

We explore the use of an N-modular redundancy-like (N-MR) approach, modeled off of existing work [4], to provide a varying amount of fault-tolerance. We create an new N-MR configuration (i.e., a new NN topology) from a non-redundant one in the following way:

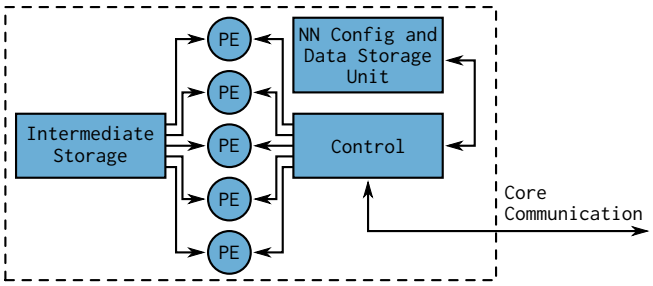

Figure 1: Our NN accelerator architecture executes NN Requests received from general-purpose cores by dynamically scheduling neurons on processing elements (PEs).

1. Hidden neurons and their weights are replicated N times
2. Weights of replicated neurons are multiplied by $1/N$

This differs from standard modular redundancy [4] as no explicit voting is used. This approach provides gradual improvements in fault-tolerance at the cost of power–performance trade-offs. We evaluate these trade-offs using an NN accelerator of our own design when executing NNs from modern applications to determine the viability of this N-MR approach.

2. Architecture

Figure 1 shows our NN accelerator architecture. This accelerator can compute one or more simultaneous multilayer perceptron (MLP) *NN Requests* from connected CPU cores. We define an NN Request as a request by a thread to compute the output of a specific NN. NN Requests are recorded in an NN Table and their configurations are cached locally after being read from memory. Control logic dynamically maps neurons in valid NN Requests to unallocated Processing Elements (PEs). The outputs of intermediate computations (from hidden neurons) are stored in Intermediate Storage.

3. Evaluation

We compute the power–performance and accuracy data for a 16-PE version of our NN accelerator placed-and-routed in a 45nm predictive technology model standard cell library [8, 3]. Our accelerator consumes 571mW at 89MHz. For selected function approximation [7] and state prediction [9] applications that use MLP NNs (see Table 1), we compute their latency and application-specific output accuracy for varying amounts of N-MR in the presence of single PE faults. We approximate a PE fault by setting the internal PE accumulator to a random value. We do not train the NNs in any special way to impart fault-tolerant properties.

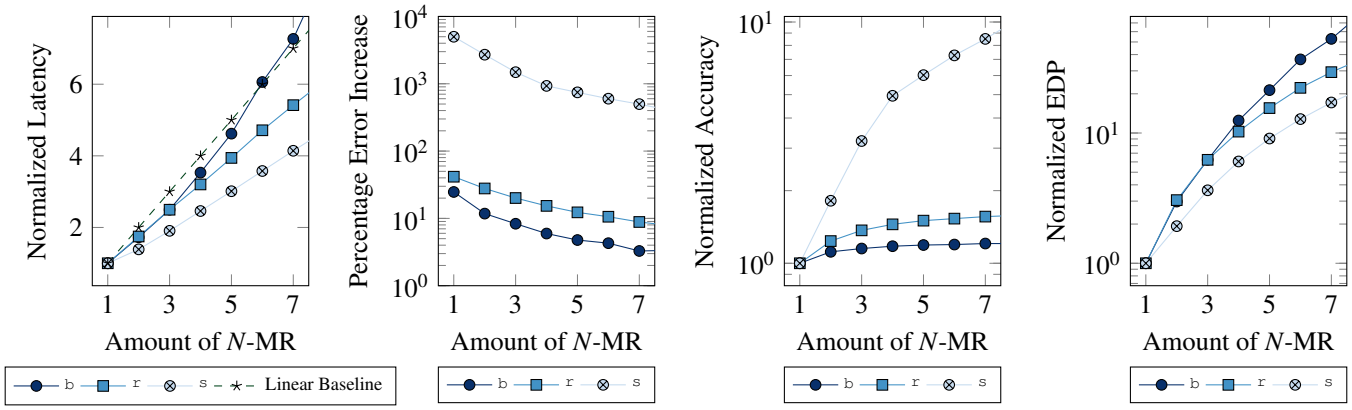

Figure 2: Normalized latency (*Left*), percentage error increase (*Mid-Left*), normalized accuracy (*Mid-Right*), and normalized energy–delay product (*Right*) for `blackscholes` (`b`), `rsa` (`r`), and `sobel` (`s`).

Table 1: Evaluated neural networks and their topologies

Application	NN Topology	Description
`blackscholes (b)` [7]	$6\times8\times8\times1$	Financial option pricing
`rsa (r)` [9]	$30\times30\times30$	Brute-force prime factorization
`sobel (s)` [7]	$9\times8\times1$	3×3 Sobel filter

Figure 2 shows normalized latency/accuracy, percentage accuracy difference compared to a fault-free NN, and normalized energy–delay product (EDP) when executing the NNs shown in Table 1 with varying amounts of N-MR on our NN accelerator. We do not report energy as it is proportional to latency.[1] We compute each accuracy datapoint as an expected accuracy when a random fault is introduced at run-time. This accuracy is normalized to the accuracy of an NN with a single fault and no redundancy. Accuracy is measured using mean squared error (MSE) for `blackscholes` and `sobel` and using the percentage of correct predictions for `rsa`. We also show accuracy compared to a fault-free NN to give a sense of scale.

NN size is a dominant factor for accuracy measurements. In small NNs (e.g., `sobel`) the individual contribution of each neuron is much higher than in large NNs (e.g., `blackscholes`, `rsa`) where computational importance is more distributed. Consequently, applying N-MR results in substantial accuracy increases for `sobel` while moderate gains are realized for `blackscholes` and `rsa`. These gains are tempered by the large percentage accuracy difference of `sobel` versus a fault-free NN. The relationship between latency/energy and redundancy is, at worst, linear, e.g., an application with no redundancy takes C cycles and uses E energy while an N redundant version takes approximately NC cycles and uses NC energy. However, as seen in the left of Figure 2, the increased work from added redundancy allows energy and latency to, generally, scale more slowly than the baseline worst case.

N-MR shows increases in normalized EDP as it increases both energy and latency. This can be seen in the right plot of Figure 2.

[1] Due to our use of a fixed power value for our 16-PE accelerator

4. Conclusion

The graph structure of NNs enables N-MR to be easily utilized to improve fault-tolerance at run-time. One simple NN configuration can be scaled up or down with dedicated multipliers and modifications to the control logic of our NN accelerator. The amount of redundancy can then be varied in accordance with the required output accuracy of a specific application or increased over time to extend the operational lifespan of a device. Nevertheless, this approach, since it uses redundancy does incur EDP costs. However, this preliminary work does demonstrate that the topology of NNs can be leveraged to provide varying levels of fault-tolerance and warrants further exploration.

5. Acknowledgments

This work was supported by a NASA Office of the Chief Technologist's Space Technology Research Fellowship.

References

[1] C.-T. Chiu *et al.*, "Modifying training algorithms for improved fault tolerance," in *WCCI*, vol. 1. IEEE, 1994, pp. 333–338.

[2] R. D. Clay and C. H. Sequin, "Fault tolerance training improves generalization and robustness," in *IJCNN*, vol. 1. IEEE, 1992, pp. 769–774.

[3] S. I. Initiative, "Nangate open cell library," Available: www.si2.org/openeda.si2.org/projects/nangatelib, 2010.

[4] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 446–456, 1995.

[5] B. E. Segee and M. J. Carter, "Comparative fault tolerance of parallel distributed processing networks," *IEEE Trans. Comput.*, vol. 43, no. 11, pp. 1323–1329, 1994.

[6] C. H. Sequin and R. Clay, "Fault tolerance in artificial neural networks," in *IJCNN*. IEEE, 1990, pp. 703–708.

[7] R. St. Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," in *ISCA*, 2014, pp. 505–516.

[8] J. Stine *et al.*, "Freepdk: An open-source variation-aware design kit," in *MSE*, June 2007, pp. 173–174.

[9] A. Waterland *et al.*, "Asc: Automatically scalable computation," in *ASPLOS*. ACM, 2014, pp. 575–590.